# Guida allo ZX Microdrive e all'Interface 1

A. Pennell

# Guida allo ZX Microdrive e all'Interface 1

# Guida allo ZX Microdrive e all'Interface 1

A. Pennell

McGRAW-HILL Book Company GmbH

**Amburgo** · New York · St Louis · San Francisco · Auckland · Bogotá · Città del Guatemala · Città del Messico · Johannesburg · Lisbona · Londra · Madrid · Montreal · Nuova Delhi · Panama · Parigi · San Juan · San Paolo · Singapore · Sydney · Tokyo · Toronto

Ogni cura è stata posta nella creazione, realizzazione, verifica e documentazione dei programmi contenuti in questo libro. Tuttavia né l'Autore né la McGraw-Hill Book Co. possono assumersi alcuna responsabilità derivante dall'implementazione dei programmi stessi, né possono fornire alcuna garanzia sulle prestazioni o sui risultati ottenibili dal loro uso, né possono essere ritenuti responsabili di danni o benefici risultanti dall'utilizzo dei programmi. Lo stesso dicasi per ogni persona o società coinvolta nella creazione, nella produzione e nella distribuzione di questo libro.

Titolo originale: *Master your ZX Microdrive* - Sunshine Books, London
Copyright © 1983 Andrew Pennell

Copyright © 1984 McGraw-Hill Book Co. GmbH
Lademannbogen 136
D 2000 Hamburg 63, RFT

I diritti di traduzione, di riproduzione, di memorizzazione elettronica e di adattamento totale e parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche), sono riservati per tutti i paesi.

Realizzazione editoriale: EDIGEO snc, via Ozanam 10a, 20129 Milano
Traduzione: Giuseppe Zappalà
Grafica di copertina: Valentina Boffa
Composizione e stampa: Litovelox, Trento

ISBN 88-7700-013-9

1ª edizione Novembre 1984

ZX Spectrum, ZX Interface 1, ZX Microdrive sono marchi registrati della *Sinclair Research Ltd.*

# Indice

# Introduzione

Questo libro cerca di illustrare i modi migliori per usare la *ZX Interface 1*, con particolare riguardo al *Microdrive*. L'interfaccia aumenta enormemente l'utilità dello Spectrum e i Microdrive offrono una memoria di massa veloce ad un costo contenuto.
Sono chiaramente spiegate sia le caratteristiche standard che molte altre "nascoste" che rendono l'interfaccia ancora più utile.
Ho incluso varie routine in codice macchina che aumentano la potenzialità del sistema, curando di presentarle in maniera da renderne l'introduzione e l'uso accessibili anche a chi non conosce la programmazione in codice macchina. Ho destinato ai lettori più esperti una sezione sulle routine della ROM dell'interfaccia e sul loro uso, nonché i listati commentati delle routine.
I programmi in codice macchina sono forniti sotto forma di una serie di numeri contenuti in istruzioni DATA e rilocati da un loop FOR-NEXT tramite dei POKE. Ho cercato, nei limiti del possibile, di svincolare il codice dalla posizione di lavoro, in modo tale da farlo funzionare in qualunque zona della memoria. Il posto migliore per il codice macchina è al di sopra della RAMTOP, che può essere abbassata tramite un CLEAR. È anche possibile sfruttare l'area riservata ai caratteri grafici definiti dall'utente (168 byte) e il buffer di stampa (256 byte). Per maggiore semplicità d'uso, ho fornito due locazioni consigliate per ogni routine, (per macchine da 16K e da 48K) in modo da permettere la contemporanea presenza in memoria di tutte le routine presentate in questo libro. Ricordatevi di abbassare opportunamente la RAMTOP prima di caricare le routine; CLEAR 32009 per 16K o CLEAR 64779 per 48K riserveranno abbastanza spazio.
Poiché alcune delle routine più lunghe contengono molti numeri, suscet-

tibili di errore, ogni routine forma una *checksum* durante la rilocazione. Se il totale non è corretto, la routine si fermerà visualizzando un opportuno messaggio. Occorrerà allora ricontrollare i dati, apportando le necessarie correzioni.
Una parte del libro tratta i metodi di protezione dei programmi e l'aggiunta di nuovi comandi al BASIC; le appendici contengono altre informazioni tecniche e documentano i noti problemi connessi all'uso dell'interfaccia.

*Andrew Pennell*

**Capitolo**

# Canali logici e fisici

# 1

Prima di illustrare le molte caratteristiche che l'Interface 1 aggiunge allo Spectrum, è necessario chiarire il concetto di canali "logici" e canali "fisici", essendo questi fondamentali per un uso efficiente dei Microdrive, della RS232 e della rete locale.
Un canale fisico è una parte del computer a cui trasmettere dati, o da cui riceverli. Tipici canali fisici sono la tastiera per l'immissione dei dati e lo schermo per la visualizzazione. I dati vengono trasmessi ai canali fisici attraverso canali logici (*stream*).
Sullo Spectrum sono disponibili dal BASIC 16 canali logici, quattro dei quali vengono inizialmente assegnati a determinati canali fisici, come mostra la tabella sottostante:

| Canale logico | Identificatore di canale | Uso come output | Uso come input |
|---|---|---|---|
| #0 | "K" | parte inferiore dello schermo | tastiera |
| #1 | "K" | parte inferiore dello schermo | tastiera |
| #2 | "S" | parte superiore dello schermo | nessuno |
| #3 | "P" | stampante ZX | nessuno |

I canali logici 0 e 1 sono identici ed entrambi connessi al canale fisico "K". Questo manda i caratteri alla parte inferiore dello schermo (dove vengono visualizzati i messaggi di errore e quanto digitato) e riceve dati

dalla tastiera. Il canale logico 2 è collegato al canale fisico "S" il quale manda i caratteri alla parte superiore dello schermo, mentre il canale logico 3 è collegato al canale fisico "P", che manda i caratteri alla stampante ZX. Il sistema operativo dello Spectrum non permette l'input dei dati dai canali logici 2 e 3. Per esempio, il comando INPUT#3;a$ genererà il messaggio di errore INVALID I/O DEVICE. Il simbolo che identifica il canale logico è il "#" (SHIFT 3) e può essere aggiunto ai comandi di PRINT, INPUT e ad alcuni altri. Per esempio, per visualizzare caratteri sulle ultime due linee dello schermo, che non sono normalmente disponibili, potete usare i canali logici 0 o 1:

```
PRINT #0; AT 0,0; "Linea 23"; AT 1,0; "Linea 24";: PAUSE 0
```

Il comando PAUSE è necessario per impedire l'emissione del messaggio "OK", che cancellerebbe quanto appena visualizzato.
Emettendo caratteri sul canale "K", è possibile che la parte inferiore dello schermo debordi in quella superiore. Si può evitare questo sfruttando il comando AT e terminando ogni testo con un punto e virgola.
Il comando PRINT #2; è analogo al solito PRINT, che emette i caratteri sul canale "S", cioè sulla parte superiore dello schermo. PRINT #3; è analogo a LPRINT, che stampa i caratteri sulla stampante ZX (canale fisico "P"). L'uso dei canali logici 2 e 3 nei programmi permette di sfruttare una variabile per determinare se i caratteri andranno inviati allo schermo o alla stampante. Il seguente programma dimostra l'uso di questa tecnica:

```
1000 INPUT "Stampante (S/N)";a$
1010 LET c=2: IF a$="s" OR a$="S" THEN LET c=3
1020 PRINT #c;"Questo è lo ZX Spectrum"
```

È anche possibile usare più canali logici nello stesso comando PRINT:

```
2000 PRINT #2;"Lo schermo";#3;"La stampante"
```

È possibile usare i comandi INPUT e LIST anche insieme ai canali logici. Con lo Spectrum in versione base (cioè senza Interface 1 connessa) sono ammessi solo i comandi INPUT #0; e INPUT #1; che sono equivalenti ai normali comandi INPUT. Comunque il comando INPUT # è estremamente utile, quando usato insieme all'interfaccia, come vedremo nei capitoli successivi.
Il comando LIST #n, dove n è un numero di canale logico, può essere usato per dirigere il listato di un programma BASIC su un canale logico prescelto. LIST #3 è equivalente a LLIST, che produce un listato sulla stampante ZX e LIST #0 produce un interessante, ma praticamente inu-

tile, effetto sullo schermo.
Il comando INKEY$ #n è utilizzabile anche con i canali logici, ma è assolutamente inutile se usato con i canali standard. INKEY$ #0 e INKEY$ #1 producono come risultato stringhe vuote e #2 e #3 non sono ammessi. I capitoli successivi mostreranno l'uso di questo comando con i canali aggiunti dall'interfaccia.

## OPEN # e CLOSE #

Con lo Spectrum standard, l'uso del comando OPEN # è piuttosto limitato. Il suo scopo è abbinare un canale fisico ad un canale logico, secondo la sintassi

```
OPEN #n,f$
```

dove n è il numero di canale logico e f$ è l'identificatore di canale. Notate che il simbolo # è parte della parola chiave OPEN #, e si ottiene in *extended mode* premendo SHIFT 4. Il valore di n può variare da 0 a 15 (in caso contrario verrà emesso un messaggio INVALID STREAM), e l'identificatore del canale deve essere una singola lettera (tanto maiuscola che minuscola), che specifichi un canale valido, cioè "K", "k", "S", "s", "P", "p". L'errata scelta dell'identificatore di canale produrrà il messaggio INVALID FILE NAME. Con l'Interface 1 connessa, saranno anche accettati gli identificatori "M", "m", "N", "n", "B", "b", "T", "t", che rappresentano i Microdrive, la rete locale, l'interfaccia RS232 in modo binario e l'interfaccia RS232 in modo testo. Questi nuovi identificatori di canale permettono l'uso del punto e virgola come separatore nel comando OPEN # ed alcuni richiedono anche di essere seguiti da dati. Ulteriori dettagli saranno dati in seguito.
Ad esempio,

```
OPEN #3;"s"
```

dirigerà tutto l'output del canale logico 3, normalmente connesso alla stampante ZX, sul canale "S", cioè la parte superiore dello schermo: LLIST produrrà pertanto il listato sullo schermo. Questo può essere utile nella correzione dei programmi che usano la stampante ZX, quando questa non è collegata, o per risparmiare carta. Il comando opposto è

```
OPEN #2,"P"
```

che dirige alla stampante tutto l'output del canale logico 2, normalmente

collegato allo schermo. L'uso contemporaneo di entrambi questi comandi può creare qualche confusione!
Potete inoltre sfruttare OPEN # con i comandi logici addizionali. Normalmente i canali logici da 4 a 15 non sono abbinati ad alcun canale fisico ed ogni tentativo di usarli produrrà il messaggio di errore INVALID STREAM. Comunque, questi canali logici possono essere aperti nel modo classico, ad esempio:

```
OPEN #4,"s"
PRINT #4;"Buona sera"
```

visualizzerà sullo schermo le parole "Buona sera". Tramite l'interfaccia, i canali logici da 4 a 15 sono usati estensivamente con le varie periferiche Sinclair.
Per annullare il comando OPEN #n, deve essere usato il suo complemento, CLOSE #n, dove n identifica, come al solito, il numero del canale logico. La chiusura dei canali logici da 0 a 3 li riporterà al loro stato iniziale, mentre la chiusura dei canali logici da 4 a 15 li riporterà allo stato di non connessione ad alcun canale fisico.
Fate attenzione al comando CLOSE # con i canali logici da 4 a 15, se non avete l'interfaccia collegata. Se il canale logico è già chiuso, a causa di un errore nel BASIC, in genere, viene emesso un singolare messaggio di errore e, talvolta, il computer può cancellare totalmente la RAM! Il problema non si pone se l'Interface è connessa: essa provvederà a correggere l'errore.

## CLEAR# e MOVE

È disponibile un altro comando usabile insieme ai canali logici: CLEAR #, che non ha niente in comune con il normale comando CLEAR. Il suo scopo è la chiusura dei canali logici da 4 a 15 ed il reset alle condizioni iniziali dei canali logici da 0 a 3. Bisognerà prestare attenzione nell'usarlo con i canali fisici aggiunti dall'interfaccia.
L'ultimo comando usabile insieme ai canali fisici è MOVE la cui sintassi può assumere varie forme, del tipo

```
MOVE #a TO #b
```

dove a e b sono numeri di canali logici e TO è una parola chiave. Questo comando legge un carattere dal canale logico a e lo emette sul canale logico b. Il processo è ripetuto fino al verificarsi di una determinata condizione, che dipende dal tipo di canale fisico cui è connesso il canale logi-

co a. Si tratta di un comando estremamente potente, le cui applicazioni saranno spiegate nei capitoli successivi.
È possibile, operando in codice macchina, creare canali logici particolari per usi speciali. Per esempio, il sistema operativo del BASIC usa internamente il canale "R" sul canale logico −1 (che non è disponibile dal BASIC) per emettere caratteri nell'area di lavoro interna. È inoltre possibile modificare i canali fisici esistenti: tutte le interfacce per stampante tipo *Centronics*, modificano il canale "P", in modo che tutti i caratteri emessi sul canale logico 3 (ad esempio da un comando LPRINT) vengano passati all'interfaccia aggiunta invece che alla stampante ZX.
Se si usano contemporaneamente vari canali logici, può essere difficoltoso tenere traccia dei canali fisici connessi ai singoli canali logici, nonché degli eventuali canali logici non usati. Il seguente programma produce una tabella riassuntiva di tutti i parametri relativi ai vari canali logici, inclusi quelli da −3 a −1, che sono disponibili solo ai programmi in codice macchina. Il programma produce una tabella sull'uso di ogni singolo canale logico che indica se possa essere sfruttato per ingresso, per uscita, o per entrambi. Completato l'elenco dei canali logici, viene richiesto un numero, la cui immissione vi fornirà maggiori dettagli sul particolare canale logico da esso identificato. Il programma, così com'è, è in grado solo di fornire informazioni relative ai canali normali. Ulteriori caratteristiche verranno aggiunte in seguito.

**Programma TABELLA DEI CANALI 1**

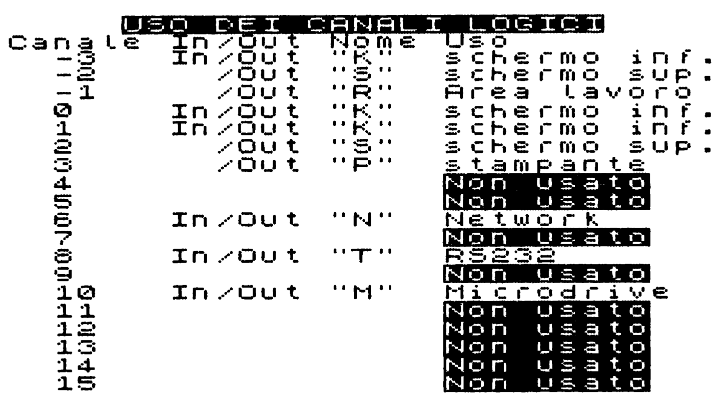
```
1000 DEF FN p(p)=PEEK p+256*PEEK (p+1)
1005 CLS : PRINT TAB 5; INVERSE 1;"USO DEI CANALI LOGI
CI"
1010 PRINT "Canale";TAB 7;"In/Out";TAB 14;"Nome";TAB 1
9;"Uso"
1020 FOR s=-3 TO 15
1025 PRINT TAB 2;s;
1030 LET d=FN p(s*2+23566+8)
1040 IF d=0 THEN  PRINT TAB 19; INVERSE 1;"Non usato":
 GO TO 1200
1050 LET d=d+FN p(23631)-1
1060 IF FN p(d+2)<>5572 THEN  PRINT TAB 7;"In";
1070 PRINT TAB 9;"/";
1080 IF FN p(d)<>5572 THEN  PRINT TAB 10;"Out";
1090 LET f$=CHR$ PEEK (d+4)
1100 PRINT TAB 14;"""";f$;"""";TAB 19;
1110 IF f$="K" THEN  PRINT "schermo inf.": GO TO 1200
1120 IF f$="S" THEN  PRINT "schermo sup.": GO TO 1200
1130 IF f$="P" THEN  PRINT "stampante": GO TO 1200
1140 IF f$="M" THEN  PRINT "Microdrive": GO TO 1200
1150 IF f$="N" THEN  PRINT "Network": GO TO 1200
```

```
1170 IF f$="T" THEN  PRINT "RS232": GO TO 1200
1180 IF f$="R" THEN  PRINT "Area lavoro": GO TO 1200
1190 PRINT  FLASH 1;"Non specificato"
1200 NEXT s
1210 PRINT AT 0,0;
1500 INPUT "Digita il numero di canale logi-co o ENTER
 per finire "; LINE S$
1505 IF S$="" THEN  STOP
1510 CLS
1515 LET S=VAL S$
1520 PRINT TAB 10; INVERSE 1;"CANALE LOGICO ";S''
1530 LET D=FN P(S*2+23566+8)
1540 IF D=0 THEN  PRINT "Canale logico chiuso": GO TO
1500
1550 LET D=D+FN P(23631)-1
1560 LET F$=CHR$ PEEK (D+4)
1570 PRINT "Identificatore di canale:";F$
1580 REM Controlla il tipo
1600 IF F$="K" THEN  GO TO 2000
1610 IF F$="S" THEN  GO TO 2020
1620 IF F$="P" THEN  GO TO 2040
1660 IF F$="R" THEN  GO TO 2050
1670 PRINT  FLASH 1;"Identificatore non conosciuto"
1700 GO TO 1500
1800 PRINT "Routine Output   :";FN P(D)
1810 PRINT "Routine Input    :";FN P(D+2)
1820 IF FN P(D)=5572 THEN  PRINT  FLASH 1;"Solo Input"
1830 IF FN P(D+2)=5572 THEN  PRINT  FLASH 1;"Solo Outp
ut"
1870 RETURN
1999 REM Canale K
2000 PRINT  INVERSE 1;"Schermo inf./tastiera"
2010 GO TO 2060
2019 REM Canale S
2020 PRINT  INVERSE 1;"Schermo sup."
2030 GO TO 2060
2039 REM Canale P
2040 PRINT  INVERSE 1;"Stampante ZX"
2045 GO TO 2060
2049 REM Canale R
2050 PRINT  INVERSE 1;"Area di lavoro"
2060 GO SUB 1800
2070 GO TO 1500
```

Il programma esegue il PEEK di varie locazioni di memoria.
Il loop di linea 1020 scandisce successivamente ogni singolo canale logico. La variabile d diventa l'offset nell'area STRMS (1030) del canale logico selezionato, che sarà pari a 0 nel caso il canale logico sia chiuso (1040). L'offset viene poi aggiunto alla variabile di sistema CHANS (1050),

con la variabile d sfruttata come puntatore a un numero di byte nella area di memoria CHANS. I canali aperti (K, S, P, R) occupano ciascuno un minimo di 5 byte nell'area di memoria CHANS. In ogni sezione di 5 byte, i primi 2 puntano ad una routine di emissione carattere, i successivi due ad una routine di acquisizione carattere ed il quinto byte è l'identificatore di canale fisico.
Alla locazione 5572 è presente la chiamata alla routine di errore INVALID I/O DEVICE (provate a digitare RANDOMIZE USR 5572); quest'indirizzo viene controllato nelle linee 1060 e 1080. L'identificatore di canale viene emesso alla linea 1100 e controllato per i canali fisici noti nelle linee 1110-1180, che emettono il nome dei dispositivi abbinati. Dopo l'elenco, l'utente può digitare un numero di canale logico (1500), su cui riceverà maggiori informazioni. Per i canali fisici correnti, queste informazioni consistono solo nelle locazioni di input e output, ottenute tramite una subroutine in 1800; successive versioni del programma forniranno maggiori dettagli. La figura mostra un esempio di output generato dal programma.

**Output del programma TABELLA DEI CANALI 1**




# Canale logico 14-z$

La seguente routine in codice macchina inizializza il canale logico 14 in modo da fargli emettere caratteri in una variabile a stringa del BASIC. Alla chiamata della routine, tramite un GO SUB 8000, la variabile st dovrà contenere l'indirizzo di memoria a partire dal quale dovrà essere rilocata la routine. Essa è lunga 101 byte ed è consigliabile che lavori a partire da 65260 (versione 48K) o da 32490 (versione 16K). Dopo la chiamata alla routine, ogni carattere emesso dal canale logico 14 sarà aggiunto alla

fine della variabile a stringa z$ (che non deve essere dimensionata). Se z$ non esiste, o è stata preventivamente dimensionata, sarà generato un errore VARIABLE NOT FOUND. È possibile usare PRINT # 14 e LIST # 14, come vedremo nel capitolo seguente in cui immagazzineremo in una stringa il catalogo di un Microdrive. Esiste tuttavia una restrizione: comandi che emettono direttamente le stringhe non lavoreranno correttamente. Per esempio

PRINT # 14;"stringa"

aggiungerà "ssssss" alla fine di z$. Per evitare questo errore, fate stampare ogni singolo carattere individualmente (PRINT # 14;"s";"t";"r";"i";"n"; "g"; "a") o usate un'altra variabile a stringa (LET a$="stringa": PRINT # 14;a$). La causa di questo comportamento è che la routine, spostando in su sezioni di memoria per inserire i caratteri, sposta in su anche l'area di lavoro dove viene immagazzinata la stringa temporanea; in questo modo viene sempre aggiunto il primo carattere.

**Programma CANALE LOGICO 14-z$**

```
7995 REM ***********************
7996 REM * canale logico 14-z$ *
7997 REM ***********************
7998 REM st=locazione iniziale
7999 REM preferibilmente:65260 / 32490
8000 RESTORE 8070
8005 LET c=0
8010 FOR i=st TO st+100
8020 READ a: POKE i,a: LET c=c+a
8030 NEXT i
8035 IF c<>10557 THEN  PRINT "Errore di checksum": STOP
8040 CLOSE #14
8050 RANDOMIZE USR st
8060 RETURN
8070 DATA 42,83,92,43,197,229,1,11
8075 DATA 0,205,85,22,209,33,59,0
8080 DATA 193,9,213,235,115,35,114,35
8085 DATA 235,1,247,255,9,1,9,0
8090 DATA 237,176,225,35,237,75,79,92
8095 DATA 167,237,66,34,50,92,1,0
8100 DATA 0,201,196,21,90,40,0,40
8105 DATA 0,11,0,245,42,75,92,126
8110 DATA 254,90,40,11,254,128,202,112
8115 DATA 6,205,184,25,235,24,240,35
8120 DATA 78,35,70,3,197,229,9,205
8125 DATA 82,22,35,235,225,193,112,43
8130 DATA 113,241,18,167,201
```

Capitolo

# I Microdrive 2

## Come funzionano

Una cartuccia per Microdrive contiene un anello chiuso di circa 5 metri di sottile nastro magnetico, di una qualità superiore rispetto a quella usata nelle normali cassette audio. I Microdrive consistono essenzialmente di un motore per generare il movimento del nastro e di una testina di registrazione/ascolto, come in un mangianastri. Il nastro viene trascinato ad alta velocità davanti alla testina e può immagazzinare dati in fase di registrazione o riprenderli in fase di lettura. Si tratta di un sistema a cassetta estremamente veloce, senza l'inconveniente di dover riavvolgere il nastro.
Il sistema ad anello chiuso ha ovviamente anche degli svantaggi: il principale è il tempo di accesso. Se la parte di nastro con il programma che si desidera caricare è appena passata davanti alla testina, sarà necessario lo scorrimento di tutto il nastro prima di riavere disponibile la sezione che ci interessa, poiché non è possibile il riavvolgimento. Questo vuol dire che possono essere necessari anche 7 secondi per trovare un programma o alcuni dati sul nastro. Anche la scrittura di dati su una di queste cartucce può essere abbastanza lenta. Se la cartuccia è praticamente nuova, con pochi altri programmi, il computer trova rapidamente una sezione libera su cui scrivere. Comunque, se una cartuccia è ben usata, il computer dovrà effettuare la ricerca di spazio su tutto il nastro, immagazzinando a pezzetti il vostro programma in posti diversi; operazione, questa, che può richiedere un certo tempo. Un programma così salvato avrà, ovviamente, un caricamento più lento.

## Protezione contro la sovrascrittura

Le audiocassette hanno due piccole linguette sul retro, che possono essere spezzate per evitare l'accidentale cancellazione di quanto registrato. Si chiamano linguette di protezione contro la sovrascrittura e sono usate, per uno scopo analogo, anche sulle cartucce del Microdrive Sinclair.
Ogni cartuccia ha su un lato una piccola linguetta di plastica. Se, servendovi di un cacciavite o di un oggetto simile, la asportate, impedirete la scrittura di dati su quella cartuccia. Questo è molto importante per i programmi commerciali e può esser utile per proteggere programmi che non devono essere cancellati. Se, in seguito, volete riutilizzare la cartuccia, vi basterà porre un piccolo pezzo di nastro adesivo sopra il taglio. Il nastro adesivo non dovrà debordare sugli altri lati della cartuccia, che potrebbe, in tal caso, non adattarsi correttamente alla sua sede.

## Precauzioni nel maneggiare le cartucce

Il nastro contenuto nelle cartucce è molto delicato. Non toccatelo mai con le dita o qualunque altra cosa e rimettete negli appositi contenitori le cartucce non in uso.
Inserendo le cartucce nei Microdrive, assicuratevi di rispettare il corretto orientamento (con l'etichetta grande in su) e spingetele a fondo. Prestate molta attenzione al completo inserimento della cartuccia, per evitare errori di lettura e/o scrittura. Occorre seguire due importanti regole:

1. Non rimuovere mai una cartuccia quando la spia rossa è accesa
2. Non accendere o spegnere il sistema se una cartuccia è presente nel Microdrive.

Il mancato rispetto di queste regole può causare un danno permanente alla cartuccia o al Microdrive.
Le cartucce durano meno delle cassette, a causa delle loro particolari condizioni operative. Conservate sempre una copia di sicurezza dei programmi o dei dati importanti su un'altra cartuccia o, meglio ancora, su una cassetta.

## Formattazione delle cartucce

Il nastro contenuto in una cartuccia vergine conterrà informazioni magnetiche casuali. Ogni tentativo di usarla produrrà il messaggio di errore

MICRODRIVE NOT PRESENT. È necessario predisporre all'uso la cartuccia tramite il comando FORMAT. Per il Microdrive, esso assume la struttura:

FORMAT "m"; d; "nome"

in cui d rappresenta il numero del Microdrive (da 1 a 8) e "nome" è il titolo, lungo fino a 10 lettere, che sarà assegnato alla cartuccia. L'operazione richiederà circa 30 secondi, durante i quali il computer inizializzerà la cartuccia, vi scriverà il titolo e collauderà, varie volte, ogni sezione del nastro. Se trova una sezione non utilizzabile perché danneggiata, contrassegnerà lo spazio in modo da evitarne il successivo uso. Durante l'operazione, lo schermo lampeggerà. Prima di avviare la formattazione, assicuratevi che la cartuccia sia correttamente inserita nel Microdrive.
Usando cartucce nuove di zecca, è consigliabile eseguire due formattazioni in rapida successione.
Il comando FORMAT cancellerà ogni programma o dato presente sulla cartuccia; prestate quindi la massima attenzione a ciò che fate. Il tentativo di formattare una cartuccia priva della linguetta di protezione genererà un errore.
Dopo la formattazione potete controllare quanto avvenuto sfruttando il comando CAT d (d è il numero del Microdrive) che, dopo circa 7 secondi, visualizzerà il titolo della cartuccia ed un numero: questo è il numero di kilobyte (K) disponibili sulla cartuccia, che dopo la formattazione dovrebbe essere pari ad almeno 85K. Il massimo che ho raggiunto su una cartuccia è 92K.

## I programmi e i Microdrive

Uno dei vantaggi principali dei Microdrive è la loro grande velocità nel salvataggio e caricamento dei programmi. Per usarli, sono necessari comandi simili a quelli sfruttati per la gestione delle cassette; ad esempio, per salvare un programma su cassetta dovrete digitare:

SAVE "canali"

Per sfruttare questo comando con il Microdrive dovrete aggiungere alcune informazioni supplementari dopo il comando. Per prima cosa un asterisco, poi "m" per segnalare che si tratta di un'operazione su Microdrive, quindi d per identificare il drive in uso. Per salvare il programma sulla cartuccia presente nel drive 2 dovrete pertanto digitare:

SAVE *"m";2;"canali"

(Potete usare indifferentemente "m" o "M"). I comandi addizionali non saranno accettati dallo Spectrum privo della Interface 1.
Ad ogni operazione del Microdrive il motore si accenderà insieme alla spia rossa. Cercando di azionare un Microdrive che non contenga l'opportuna cartuccia, sentirete uno stridio e vedrete visualizzato il messaggio di errore MICRODRIVE NOT PRESENT. Potete essere in grado di capire il corretto funzionamento di una combinazione drive/cartuccia, dal rumore che emettono. In condizioni di normale funzionamento dovreste udire un lieve ronzio. Se qualcosa non funziona, ad esempio la cartuccia non è correttamente inserita, il drive farà uno strano rumore. In tal caso, spingete a fondo la cartuccia nel drive; se il rumore continua, interrompete l'operazione ed esaminate la cartuccia, sostituitela e riprovate. Se il drive emette dei cigolii quando la cartuccia è inserita e viene visualizzato un messaggio di errore, è possibile che la cartuccia sia difettosa e che il nastro non possa girare correttamente. Se tutto è a posto, durante l'operazione di SAVE, il drive dovrà funzionare per almeno 7 secondi, durante i quali verrà ricercato il nome del file prescelto. Se ne viene trovato uno, sarà visualizzato il messaggio di errore WRITING TO A 'READ' FILE. Il lampeggio dello schermo indicherà l'operazione di scrittura dati sulla cartuccia.
Analogamente a quanto avviene con le cassette, è possibile predisporre (con la funzione LINE) l'esecuzione automatica del programma. Per salvare il programma già presentato in modo che la sua esecuzione cominci automaticamente dalla linea 1000, potrete usare

```
SAVE *"m";2;"canali"LINE 1000
```

Si potrà verificare il corretto completamento dell'operazione sfruttando il comando

```
VERIFY *"m";2;"canali"
```

Nel caso di non corrispondenza fra quanto registrato sulla cartuccia e quanto presente nella memoria del computer, verrà emesso il messaggio di errore VERIFICATION FAILED. I programmi registrati su Microdrive dovrebbero sempre fornire esito positivo alla verifica; se così non fosse, controllate attentamente tanto la cartuccia che il Microdrive e se il difetto persiste, sostituiteli.
Per caricare un programma dalla cartuccia, usate il comando LOAD con la sintassi

```
LOAD *"m";d;"nome"
```

dove d è il numero del Microdrive e "nome" è il nome del programma ri-

chiesto. A differenza di quanto avviene sulle cassette, non è possibile caricare il primo programma su nastro usando come nome del file la stringa vuota. Per esempio, digitando

LOAD *"m";1;" "

otterrete il messaggio di errore INVALID FILENAME. Se il nome del programma non è presente sulla cartuccia (o non riesce ad essere letto) verrà generato il messaggio di errore FILE NOT FOUND.
È disponibile anche la funzione MERGE:

MERGE *"m";d;"nome"

Questo comando caricherà il programma "nome" dal Microdrive d e lo fonderà con il programma e le variabili presenti in memoria. A differenza di quanto avviene con le cassette, non è possibile il MERGE di un programma predisposto per l'autoesecuzione, salvato, cioè, con l'opzione LINE. Ogni tentativo di fusione di questo tipo di programma produrrà il messaggio di errore MERGE ERROR.
Un altro comando, non necessario per i file su cassette, è ERASE. Esso cancella dalla cartuccia i file, secondo la sintassi

ERASE "m";d;"nome"

A differenza degli altri comandi per il Microdrive, ERASE non richiede l'asterisco. Non è possibile far seguire al nome del file successive informazioni o comandi.
Se interrompete un'operazione di SAVE, avrete sul nastro un file incompleto e non caricabile. Dovrete cancellare (con ERASE) ogni file incompleto. Se cercate di salvare (con SAVE) un programma con un nome già presente sulla cartuccia, sarà generato un messaggio di errore WRITING TO A 'READ' FILE. Dovrete pertanto cancellare il vecchio programma prima di salvare il nuovo.

## Concatenamento dei programmi

Usando il comando SAVE *"m"...LINE è possibile l'autoesecuzione di un programma che ne carichi un altro, eventualmente in codice macchina. Il problema è che un programma così concatenato potrà funzionare solo su un particolare drive. Per rendere l'operazione indipendente dal Microdrive, il primo programma dovrà avere una linea del tipo

LET d=PEEK 23766

La variabile d conterrà quindi il numero del drive da cui caricare il programma. Se il secondo programma si chiama "prog2", per ottenere il suo caricamento potete quindi usare

```
100 LOAD *"m";d;"prog2"
```

I due programmi potranno quindi funzionare indipendentemente dal drive da cui saranno caricati. Non è possibile, per ragioni tecniche, usare

```
100 LOAD *"m";PEEK 23766;"prog2"
```

Usate sempre una variabile.

## Array e codice macchina sulle cartucce

Sulle cartucce è possibile immagazzinare array, codice macchina, byte della memoria video, semplicemente aggiungendo opportune parole chiave al nome del file nei comandi. Per immagazzinare array, fate seguire al nome del file la parola chiave DATA e il nome dell'array numerico o stringa. Per esempio, per salvare l'array b( ) sul drive 2 sotto il nome "paghe", usate

```
SAVE *"m";d;"paghe" DATA b()
```

Con le cassette, è possibile salvare su nastro le variabili stringa non dimensionate che però, quando ricaricate, vengono alterate. Questo è stato evitato con i Microdrive; infatti digitando

```
LET a$="prova":SAVE *"m";1;"stringa"DATA a$()
```

otterrete il messaggio di errore NONSENSE IN BASIC.
È possibile salvare il codice macchina e i byte aggiungendo la parola chiave CODE seguita da due numeri, separati da una virgola. Il primo numero è l'indirizzo iniziale e il secondo il numero di byte. Entrambi i numeri sono opzionali nei comandi di LOAD e VERIFY. I byte della memoria video possono essere salvati usando SCREEN$. A differenza di quanto avviene con i file su cassetta, è possibile verificare i byte salvati.
Cercando di caricare un file usando un comando errato, si otterrà un errore di tipo WRONG FILE TYPE; ad esempio,

```
LOAD *"m";1;"test" CODE
```

genererà errore se "test" è un programma.

## Il comando CAT

Questo comando serve per esaminare i file presenti su una cartuccia. Esso ha la struttura

CAT d

dove d è il numero del Microdrive. Esso produce sullo schermo il titolo della cartuccia, una linea vuota, la lista ordinata alfabeticamente di tutti i file sulla lista ed infine un numero. Il numero indica quanto spazio libero resta sulla cartuccia in Kbyte che, su una cartuccia vuota, deve essere pari ad almeno 85K. È possibile inviare il catalogo a canali fisici diversi dallo schermo usando

CAT #n,d

dove n è il numero del canale logico. Per esempio,

CAT #3,2

produrrà il catalogo del drive sul canale logico 3, normalmente la stampante ZX. Esso listerà però solo i primi 50 file trovati: per questo motivo, se avete registrato più di 50 file su una cartuccia, non potrete avere un catalogo completo.
Supponiamo ora di voler conoscere dall'interno di un programma quali file siano presenti sulla cartuccia. Un sistema piuttosto rudimentale e lento consisterebbe nel pulire lo schermo, eseguire un CAT e leggere quindi ogni singolo carattere dello schermo usando la funzione SCREEN$. Questo metodo, oltre ad essere macchinoso, è di scarsa utilità se sulla cartuccia sono presenti più di 20 file, poiché lo schermo slitterebbe in su, perdendo i nomi dei primi file. Il miglior metodo consiste nell'usare la routine presentata alla fine del Capitolo 1 per predisporre il canale logico 14 per aggiungere caratteri alla variabile z$, quindi, sfruttando

LET z$="": CAT #14,d

avrete la stringa z$ contenente l'intero catalogo, tranquillamente esaminabile. Per essere in grado di usare le informazioni ottenute da z$, dovete tenere presente la sua struttura: vi sono innanzitutto 10 caratteri che rappresentano il titolo della cartuccia, seguiti da due codici di "a capo" (CHR$ 13). Quindi i nomi dei file, ognuno composto da 10 caratteri, in ordine alfabetico, separati fra loro da un a capo. Infine vi è un altro a capo,

uno o due caratteri con il numero di Kbyte liberi ed un altro a capo. Questo schema viene reso più chiaro dalla seguente routine che produce sullo schermo una rappresentazione agevolmente leggibile. Il codice macchina della routine CANALE LOGICO 14–z$ deve essere immesso in memoria prima del GO SUB 2000.

**Routine CATALOGO ESTESO**

```
1999 REM CATALOGO ESTESO
2000 LET z$="": CAT #14,d
2010 CLS
2020 PRINT "Nome cartuccia:"; INVERSE 1;z$( TO 10)
2025 PRINT
2030 LET z$=z$(13 TO )
2040 LET f=0
2050 IF LEN z$<10 THEN  GO TO 2100
2060 LET f=f+1
2070 PRINT f;". ";z$( TO 10)
2080 LET z$=z$(12 TO )
2090 GO TO 2050
2100 PRINT ''f;"file   ";z$(2 TO LEN z$-1);"K disponib
ili"
2110 RETURN
```

## Immagazzinamento delle variabili

Supponiamo che abbiate un programma abbastanza grande, con relativamente pochi array e variabili: tanto il salvataggio dell'intero programma insieme alle variabili (con un normale SAVE), quanto il salvataggio di ogni singolo array e di ogni singola variabile con un differente nome di file (usando SAVE ... DATA) possono essere operazioni lunghe e noiose. Le seguenti subroutine vi permettono di salvare tutte le variabili di un programma su una cartuccia e di ricaricarle in seguito anche in un altro programma. Le variabili sono salvate sulla cartuccia 1 sotto i nomi "var1" e "var2", che possono essere agevolmente cambiati alterando le linee 4040, 4050, 4110 e 4130. Non è possibile, a causa della tecnica usata, passare il numero del drive e il nome dei file come variabili. Notate che le linee 4010 e 4015, pur essendo identiche, sono entrambe necessarie. La routine SAVE può essere chiamata con un GO SUB 4000, mentre la routine LOAD deve essere chiamata con un GO TO 4100. Il GO SUB non può essere usato poiché contiene un comando di CLEAR che cancella tutti i precedenti GO SUB, rendendo quindi necessario, in tal caso, variare la linea 4140 per tornare alla corretta linea del programma chiamante.

**Routine SAVE E LOAD VARIABILI**

```
3999 REM Salva le variabili
4000 DEF FN p(p)=PEEK p+256*PEEK (p+1)
4010 LET l=FN p(23641)-FN p(23627)
4015 LET l=FN p(23641)-FN p(23627)
4020 POKE 23565,l-256*INT (l/256)
4030 POKE 23566,INT (l/256)
4040 SAVE *"m";1;"var1"CODE 23565,2
4050 SAVE *"m";1;"var2"CODE FN p(23627),l
4060 RETURN
4098
4099 REM Ricarica in memoria le variabili
4100 CLEAR
4110 LOAD *"m";1;"var1"CODE 23565
4120 DIM a$(FN p(23565)-7)
4130 LOAD *"m";1;"var2"CODE FN p(23627)
4140 GO TO xxxx: REM seguito del programma
```

## Copie multiple

Come già accennato, è sempre raccomandabile mantenere copie di sicurezza di dati e programmi importanti su altre cartucce o su cassette. Lo Spectrum fornisce la possibilità (non documentata) di salvare un file più volte sulla stessa cartuccia, con lo stesso nome. Tuttavia, se dal BASIC cercate di salvare due volte lo stesso file con lo stesso nome, verrà generato un errore. Per ottenere più copie di un programma o di qualunque altro file sulla stessa cartuccia, prima di digitare il comando SAVE, è necessario eseguire un POKE 23791,x dove x è il numero di copie, da 1 a 255, che desiderate; vi consiglio di tenere due copie. All'esecuzione del comando SAVE il file sarà scritto x volte. Il numero di copie viene riportato a 1 dopo il SAVE. Questo meccanismo è invisibile all'utente, in quanto CAT visualizzerà solo una volta il nome del file anche se questo è presente in più copie. Il vantaggio di questo metodo di copie multiple è che se cancellate accidentalmente il file, cancellerete solo una delle copie presenti. Le copie rimanenti resteranno sulla cartuccia e potranno essere caricate nella maniera usuale. Ovviamente, se fate x copie di un file, la loro completa cancellazione richiederà x comandi di ERASE. Non dimenticate che le copie multiple occupano x volte più spazio sulla cartuccia e che un comando di formattazione le cancellerà tutte contemporaneamente.

Capitolo

# Gestione dei file dei Microdrive 3

## File sequenziali

Il file è un blocco di dati cui è possibile aggiungere argomenti o da cui è possibile leggere informazioni. Con i file ad accesso sequenziale è necessario leggere e scrivere i dati in un particolare ordine. Per esempio, desiderando il decimo dato, è necessario leggere i nove precedenti. Sullo Spectrum, i file sono normalmente immagazzinati su una cartuccia e sono di tipo sequenziale (gli altri tipi di file, tra cui quelli ad accesso casuale, non sono disponibili dal BASIC).
Il Capitolo 1 ha descritto il funzionamento dei canali logici e dei canali "K", "S" e "P". Per immagazzinare dati sui Microdrive, ferma restando la possibilità di salvarli come programmi o array, viene usato un altro identificatore di canale, "m" (o "M"). Analogamente, deve essere usato il comando OPEN # per abbinare il canale fisico "m" ad un canale logico; la sintassi di questo comando, quando è usato per il Microdrive, richiede più informazioni.
Essa assume la struttura

```
OPEN #n;"m";d;"nome"
```

che crea un nuovo canale identificato con la lettera "m" sul drive d chiamato "nome", e abbina questo canale fisico al canale logico "m". Questo file poiché è stato appena creato e non può quindi contenere dati, è un file di scrittura.

## File di scrittura

Si tratta di file che non esistono prima dell'esecuzione di un comando OPEN #, da cui sono creati. Il comando principale per inviare dati ad un file su Microdrive è PRINT #n, che lavora in modo analogo al normale PRINT, ma invia i caratteri alla cartuccia. Come dimostrazione, eseguite il programma seguente e prestate attenzione agli istanti in cui il Microdrive è in funzione.

```
100 OPEN #4;"m";1;"testfile"
110 FOR i=1 to 500
120 PRINT i;" ";
130 PRINT #4;i
140 NEXT i
150 CLOSE #4
```

Potreste aspettarvi che il Microdrive resti in funzione via via che ogni singolo numero viene emesso, ma non è così. Dopo il comando OPEN #, il drive resterà in funzione per un certo tempo, mentre ricerca un file chiamato "testfile", che non dovrebbe trovare. OPEN # crea un'area di memoria chiamata *buffer,* che viene usata per l'immagazzinamento temporaneo dei caratteri. Il canale PRINT # immagazzinerà i caratteri nel buffer fino al suo riempimento ma non li scriverà sulla cartuccia. A buffer pieno, il drive starà in funzione per circa un secondo durante il quale trasferirà i 512 caratteri del buffer sulla cartuccia; il buffer sarà quindi svuotato, pronto a raccogliere altri dati. Questo è il motivo per cui il drive opererà solamente un paio di istanti, mentre il buffer si riempirà circa 4 volte. Con questo tipo di file, è assolutamente necessario il comando CLOSE #, perché, se il buffer non è pieno e non eseguite questo comando scritto, gli ultimi dati che pensavate di aver scritto tramite il comando PRINT # non raggiungeranno mai la cartuccia, ed il file sarà incompleto. Il comando CLOSE # trasferirà sulla cartuccia tutti i dati presenti nel buffer, che sarà poi rimosso dalla memoria. Eseguendo il comando CLEAR # mentre è aperto un canale logico di scrittura, il file sarà incompleto ed inutilizzabile e dovrà essere cancellato. Se provate ad aprire (tramite OPEN) un file di scrittura che è già aperto su un altro canale logico, verrà visualizzato il messaggio READING FROM A 'WRITE' FILE.

## File di lettura

Un file di scrittura che sia stato creato con OPEN #, abbia ricevuto i dati e sia stato chiuso, diventa un file di lettura, visibile sul catalogo della

cartuccia. Per leggere i dati in esso contenuti, bisogna aprirlo, usando nuovamente OPEN #, con la stessa sintassi già usata. La vera e propria lettura dei dati viene generalmente eseguita con il comando INPUT #. Per leggere i dati precedentemente immessi, sfruttate il seguente programma:

```
100 OPEN #4;"m";1;"testfile"
110 INPUT #4;i
120 PRINT i;" ";
130 GO TO 110
```

Questo programma predispone come canale di lettura di file, il canale logico 4. La variabile i assume il suo valore prendendolo dal file (linea 110) e viene visualizzata sullo schermo. Il programma ritorna in ciclo, fermandosi solo quando, letto tutto il file, avverrà l'errore di END OF FILE. Vedremo in seguito come scoprire la fine del file prima di incontrare l'errore. Come potete immaginare, il programma crea un buffer per 512 caratteri. Quando viene incontrato il comando INPUT #, un blocco di 512 caratteri (o meno, se è l'ultimo blocco) viene letto dalla cartuccia ed immagazzinato nel buffer. Quando viene poi richiesto un carattere dalla routine INPUT (o INKEY$ #), esso viene prelevato dal buffer. Se si tratta dell'ultimo carattere del buffer, verrà letto dalla cartuccia un altro blocco di 512 byte finché il file non sarà stato interamente riversato, la qual cosa genererà il messaggio di errore END OF FILE.
C'è un altro sistema di leggere caratteri da un file, che sfrutta la funzione INKEY$ #, brevemente menzionata nel Capitolo 2. A differenza di INPUT, che mette insieme una sequenza di caratteri fino al raggiungimento della fine della linea, copiandoli poi in una variabile, INKEY$ #n legge un singolo carattere. A differenza della normale funzione INKEY$, essa attende un carattere: INKEY$ # usato con un canale logico Microdrive non produrrà mai una stringa vuota come risultato. Finita la lettura di un file, è opportuno terminare l'operazione con un comando CLOSE#. Anche se non necessario, è consigliabile agire in tal modo sia per buona abitudine, sia per liberare la zona di memoria precedentemente occupata dal buffer. È anche possibile eseguire un CLEAR #.
Usando il comando INPUT, il contatore sfruttato dallo Spectrum per produrre il messaggio SCROLL? viene resettato; quindi, se state leggendo un file e lo state visualizzando sullo schermo, esso scorrerà continuamente verso l'alto. Se questo effetto non è gradito, sfruttate una linea del tipo

```
LET sc=PEEK 23692: INPUT #n: POKE 23692,sc
```

Questa riga resetterà il contatore ogni volta che viene eseguita. In effetti,

un semplice sistema per disabilitare il messaggio di scroll in un programma è l'uso del comando

```
INPUT " "
```

## Note su PRINT# e INPUT#

Occorre prestare molta attenzione usando i comandi PRINT # e INPUT# con i file su Microdrive, a causa dell'effetto dei separatori usati fra gli argomenti. Visualizzando i dati sullo schermo, una virgola sposterà il cursore alla successiva semilinea ed un apostrofo alla successiva linea. Non esistono però linee nei file su Microdrive, e l'aggiunta di una virgola ad un file (ad esempio PRINT #4;a,b) manderà in effetti il codice di controllo della virgola, cioè CHR$ 6. Analogamente, un apostrofo emetterà il CHR$ 13 (il codice di a capo). Il codice di carattere 6 può confondere lo Spectrum se cercate di usare il comando INPUT # (ma non INKEY #). Provate, ad esempio, il seguente programma:

```
10 OPEN #4;"m";1;"errortest"
20 LET a$="primo"; LET b$="secondo"
30 PRINT #4;a$,b$
40 CLOSE #4
50 OPEN #4;"m";1;"errortest"
60 INPUT #4;a$
70 INPUT #4;b$
80 CLOSE #4
```

Sarete forse stupiti dall'errore END OF FILE generato dalla linea 70, ma il PRINT # in linea 30, manda effettivamente alla cartuccia il testo "primo", quindi una virgola (CHR$ 6), quindi il testo "secondo" ed infine un a capo (CHR$ 13). Il comando INPUT # suppone che le variabili nel file siano separate dal CHR$ 13, quindi legge a$ come "primo"+CHR$ 6+"secondo", svuotando così il file; ogni tentativo di leggere oltre produrrà errore. I separatori nei comandi INPUT # possono creare problemi inaspettati. Provate:

```
10 OPEN #4;"m";1;"testfile"
20 INPUT #4;a$,b$
```

(supponendo che il file "testfile" esista). Si genererà l'errore WRITING TO A 'READ' FILE. Questo è causato dalla virgola di linea 20 che cerca di mandare un CHR$ 6 ad un file di lettura. Se le virgolette (") vengono

incluse nei file di dati, la lettura delle stringhe tramite il comando INPUT # può causare problemi; sarà pertanto opportuno usare sempre INPUT #...LINE. Le regole d'oro per l'uso di PRINT # e di INPUT # sono:

1. Separate sempre le variabili con l'apostrofo quando usate PRINT # o usate linee separate: PRINT #4;a$'b$ o PRINT #4;a$: PRINT;#4b$
2. Usate sempre il punto e virgola come separatore nei comandi di input e aggiungete la parola LINE per le variabili a stringa: INPUT #4; LINE a$; LINE b$

## Uso del comando MOVE con i file su Microdrive

Il comando MOVE ha molti usi, dei quali descriveremo solo quelli correlati ai file su Microdrive. Se avete un file di dati e volete sapere cosa contiene, senza necessità di aprirlo e leggerlo carattere per carattere, potete eseguire:

```
MOVE"m";d;"nome" TO #2
```

(TO è una parola chiave), che visualizzerà il contenuto del file specificato sul canale logico 2, lo schermo. (Se provate a eseguire l'identica operazione con un altro tipo di file, come un programma, ottenete un messaggio di errore WRONG FILE TYPE). Se volete la copia su carta del file, potete sfruttare

```
MOVE "m";d;"nome" TO #3
```

che stamperà il file sul canale logico 3, la stampante ZX. Potete anche usare MOVE per copiare file di dati:

```
MOVE "m";d1;"file 1" TO "m"; d2; "file 2"
```

che copia i dati dal file 1 sul drive d1 al file 2 sul drive d2.

## Come espandere un file

Supponiamo che abbiate un file che contenga molti dati, ad esempio la lista del vostro software, e che desideriate aggiungere alla fine alcuni nuovi argomenti. Non potendo scrivere dati su un file di lettura, dovete

creare un nuovo file, copiare il contenuto del vecchio file nel nuovo e, prima di chiuderlo, aggiungere nuovi dati. Un sistema per ottenere questo consiste nel leggere ogni singolo argomento dal vecchio file e metterlo sul nuovo; questo metodo è però piuttosto lento e poco efficiente e non lavora con alcuni tipi di dati. Il modo migliore consiste nell'usare il comando MOVE come mostra il seguente esempio, in cui aggiungiamo le stringhe "invasori spaziali" e "25 dicembre 1985" al file "software":

```
10 OPEN #4;"m";1;"software2"
20 MOVE "m";1;"software" TO #4
30 PRINT #4;"invasori spaziali" "25 dicembre 1985"
40 CLOSE #4
```

Se desiderate che il nuovo file abbia lo stesso nome del vecchio, potete aggiungere le linee

```
50 ERASE "m";1;"software"
60 MOVE "m";1;"software2" TO "m";1;"software"
70 ERASE "m";1;"software2"
```

Il programma sarà eseguito molto più velocemente se i file sono su due diversi Microdrive. Il comando MOVE è anche molto utile per sommare due file di dati, ad esempio per formare il file "terzo" aggiungendo il file "secondo" al file "primo":

```
10 OPEN #4;"m";1;"terzo"
20 MOVE "m";2;"primo" TO #4
30 MOVE "m";1;"secondo" TO #4
40 CLOSE #4
```

La velocità del programma appena presentato può essere enormemente aumentata disponendo di due o anche di tre Microdrive, su ognuno dei quali sia presente un file.

## Uso del comando LIST#

Anche se il metodo più usato per scrivere dati su una cartuccia è il comando PRINT #, è possibile usare anche LIST #. Esso scriverà il listato di un programma BASIC su un canale logico, che può essere un canale di Microdrive. Rileggendo il file, ogni simbolo (ad es. STOP) sarà rappresentato da un solo carattere (CHR$ 266 per STOP), e non da singoli caratteri con spazi fra le parole chiave. La conversione di un programma in singoli

caratteri con LIST# può essere molto utile se desiderate elaborare o eseguire delle ricerche sul vostro programma BASIC, eventualmente con un *word-processor*.
Il programma seguente cercherà in un file di dati creato con OPEN #, LIST # e CLOSE # ogni particolare sequenza e visualizzerà sullo schermo ogni linea che la contenga. Questo è molto utile per rintracciare variabili o per modificare alcuni GO TO. Può essere usato anche per scandire qualunque tipo di file di dati e non soltanto i listati dei programmi. Esso può anche terminare con il messaggio END OF FILE.

```
 999 REM Programma ricerca
1000 INPUT "Numero del drive?";d,"Nome del file?";LINE f$
1005 CLOSE #4: OPEN #4;"m";d;f$
1010 INPUT "Stringa da ricercare?";LINE b$
1020 INPUT #4; LINE a$
1030 FOR i=1 TO LEN a$-LEN b$
1035 IF a$(i TO i+LEN b$-1)=b$ THEN PRINT a$: PAUSE
     50: GO TO 1050
1040 NEXT i
1050 GO TO 1020
```

Se desiderate ricercare comandi, ad esempio GO TO, dovrete battere THEN, che pone lo Spectrum nel modo K, quindi premere il tasto opportuno (G per GO TO), e cancellare la parola THEN.

## Fine del file

Durante la lettura di un file, potreste raggiungerne la fine: se cercate di leggere ulteriormente, otterrete la visualizzazione del messaggio END OF FILE. Esistono vari modi per scoprire la fine del file (EOF da END OF FILE) prima di raggiungerla, che migliorano i programmi rendendoli più efficienti.
Il modo più semplice per scoprire un EOF consiste nell'emettere un carattere speciale o una sequenza di caratteri normalmente non usati alla fine del file, prima di chiuderlo. Io generalmente sfrutto la sequenza CHR$ 0+CHR$ 0, che non uso per alcun altro scopo. Questo metodo è il migliore per gran parte dei programmi, ma non può essere usato se avete un programma ideato per leggere tutti i tipi di file di dati, o che può leggere file di programmi o di codice macchina (usando un metodo che sarà spiegato in seguito). Altri BASIC generalmente dispongono di una funzione ON EOF GO TO o più in generale ON ERROR GO TO, che cede il controllo ad una particolare linea nel caso avvenga un errore o si raggiunga

la fine del file. Purtroppo lo Spectrum non dispone di questa funzione, ma un paio di linee di BASIC possono agevolmente rilevare il 99% degli EOF. Aggiungendo le seguenti linee ad un programma (il più vicino possibile al suo inizio), la funzione FN E(x), dove x è il numero del canale logico, restituirà 1 se il file è vuoto, 0 in ogni altro caso.

```
10 DEF FN E(A)=((INT (PEEK (FN D(A)+67)/2)-2*INT
   (PEEK (FN D(A)+67)/4) AND (FN P(FN D(A)+11)>=FN P(FN
   D(A)+69))))
11 DEF FN P(P)=PEEK P+256*PEEK (P+1)
12 DEF FN D(D)=FN P(D*2+23574)+FN P(23631)-1
```

(Se il canale logico x non è aperto o se non è un canale "M", il risultato sarà privo di significato). Questa funzione non lavora correttamente quando il numero di caratteri nel buffer è un sottomultiplo di 512, eventualità fortunatamente rara. Creando un file di scrittura, è possibile assicurarsi di non predisporre la condizione di errore, usando la funzione

```
FN P(FN D(X)+11
```

un attimo prima di chiudere il canale logico x. Se questo è 0, viene inviato sul canale logico un ulteriore carattere.
In alternativa, è possibile usare la seguente routine in codice macchina per ottenere la funzione ON EOF GO TO. Essa è lunga 67 byte e può essere posizionata in qualunque zona della memoria. Prima di chiamarla tramite GO SUB 8150, occorrerà inizializzare le variabili EOF e line. La routine pone in memoria il codice e lo esegue. Fatto ciò, ogni volta che capita un EOF, non sarà visualizzato alcun messaggio e l'interprete salterà al numero di linea specificato. La funzione viene cancellata ad ogni errore, compreso EOF. Se desiderate cambiare il numero di linea a cui cedere il controllo dopo l'errore, variate ed eseguite GO SUB 8220. Verificatosi l'EOF, per trovare in quale linea ciò sia avvenuto, usate:

```
LET errline=PEEK 23753+256*PEEK 23754
```

Eseguite questo prima di qualunque operazione su Microdrive. In caso contrario otterrete un risultato non valido.

**Routine ON EOF GO TO**

```
8145 REM ********************
8146 REM *   On EOF GO TO   *
8147 REM ********************
```

```
8148 REM eof=locazione iniziale, line=linea cui saltar
e in caso di errore
8149 REM preferibilmente:65190 / 32490
8150 RESTORE 8250
8160 LET c=0
8170 FOR i=eof TO eof+66
8180 READ a: LET c=c+a
8190 IF a<>260 THEN  POKE i,a
8200 NEXT i
8210 IF c<>6456 THEN  PRINT "Errore di checksum": STOP
8220 POKE eof+49,line-256*INT (line/256)
8225 POKE eof+50,INT (line/256)
8230 RANDOMIZE USR eof
8240 RETURN
8250 DATA 33,17,0,9,235,42,61,92
8260 DATA 115,35,114,207,49,1,0,0
8270 DATA 201,42,61,92,58,58,92,254
8280 DATA 7,194,3,19,94,35,86,213
8290 DATA 205,176,22,253,203,55,174,205
8300 DATA 110,13,42,69,92,34,201,92
8310 DATA 17,260,260,33,66,92,115,35
8320 DATA 114,35,54,1,253,54,0,255
8330 DATA 195,125,27
```

## Lettura dei file di programma come file di dati

Normalmente, se cercate di aprire un canale di Microdrive con il nome di file di programma, o ogni altro tipo di file non di dati, otterrete il messaggio di errore WRONG FILE TYPE. Tuttavia può essere estremamente utile leggere questi tipi di file dal BASIC, e ciò è ottenibile sfruttando questa routine in codice macchina, chiamata OPEN#OGNI TIPO. Il suo funzionamento è pressoché equivalente al comando OPEN, ma aprirà qualunque file, indipendentemente dal suo tipo ed inoltre, cosa molto utile, fornirà informazioni sull'esistenza del file. Prima di eseguire il GO SUB 8350, dovrete inizializzare la variabile open alla locazione di lavoro per la routine (consiglio 32320 per lo Spectrum 16K e 65090 per la versione 48K). Il GO SUB porrà in memoria il codice, senza eseguirlo.

**Routine OPEN #OGNI TIPO**

```
8344 REM **********************
8345 REM *  OPEN# ogni tipo  *
8346 REM *********************
8347 REM open=locazione iniziale
8348 REM preferibilmente:65090 / 32320
```

```
8350 RESTORE 8400: LET c=0
8360 FOR i=open TO open+93
8370 READ a: LET c=c+a
8375 POKE i,a
8380 NEXT i
8385 IF c<>10725 THEN  PRINT "Errore di checksum": STOP
8390 RETURN
8400 DATA 58,216,92,205,39,23,33,17
8410 DATA 0,175,237,66,1,0,0,216
8420 DATA 50,215,92,33,10,0,34,218
8430 DATA 92,42,123,92,34,220,92,58
8440 DATA 216,92,135,33,22,92,95,22
8450 DATA 0,25,217,229,217,229,207,34
8460 DATA 221,203,24,70,40,13,175,207
8470 DATA 33,207,44,225,217,225,217,1
8480 DATA 1,0,201,221,203,4,190,175
8490 DATA 229,207,33,209,225,115,35,114
8500 DATA 221,126,67,230,4,198,2,79
8510 DATA 6,0,217,225,217,201
```

Quando desiderate eseguire il programma, dovrete comunicargli a quale drive, canale logico e file siete interessati. Potete farlo ponendo in 23766 il numero del Microdrive, in 23768 il numero del canale logico e nelle prime 10 locazioni dell'area riservata ai caratteri grafici definiti dall'utente, il nome del file. Ad esempio, se desiderate ottenere OPEN #6;"m";2;"test" (dove "test" è qualunque tipo di file), le istruzioni saranno:

```
3500 POKE 23766,2: REM numero del drive
3510 POKE 23768,6: REM numero del canale logico
3520 LET a$="test"
3530 FOR i=1 TO 10
3540 IF i>LEN a$ THEN POKE USR "a"+i-1,32: GO TO 3560
3550 POKE USR "a"+i-1, CODE a$(i)
3560 NEXT i
3570 LET a=USR open: REM chiama la routine
```

Notate che se il nome del file è lungo meno di 10 caratteri, occorrerà aggiungere un appropriato numero di spazi (CHR$ 32) come avviene in linea 3540. Il valore restituito dalla funzione USR open ("a" nel nostro esempio), è così interpretabile:

0 – canale logico già aperto
1 – file non trovato
2 – file di dati
6 – file non di dati

Se il file non esiste, il canale logico viene chiuso e il file non viene creato. Il metodo vale pertanto solo per i file di lettura; i file di scrittura non vengono creati da esso, a differenza di quanto avviene con il comando OPEN #.
Se il valore restituito dalla funzione è 6, rappresentativo di un file non di dati, occorrerà conoscere che tipo di file sia. Quest'informazione viene fornita dai primi nove caratteri letti dal file che contengono tutti i suoi attributi, incluso il tipo. Il primo carattere determina il tipo secondo la seguente tabella:

0 – programma BASIC
1 – array numerico
2 – array stringa
3 – byte

I successivi otto byte contengono dettagli come lunghezza, linea d'inizio ecc., e sono direttamente prelevati dalle variabili di sistema HD_00–HD_11 (vedi Appendice A per maggiori dettagli). Quanto segue i primi nove byte, è il file vero e proprio. Il seguente programma, CATALOGO COMPLETO, è una versione molto migliorata della funzione CAT che oltre a fornire tutti i nomi dei file presenti, comunica vari dettagli sul file (tipo, lunghezza, numero di linea di autostart, ecc.). Essa richiede le routine in codice macchina CANALE LOGICO14-z$ e OPEN #OGNI TIPO. Questo programma è più lento del CAT, ma fornisce una grande quantità di utili informazioni.

**Programma CATALOGO COMPLETO**

```
4997 REM ***********************
4998 REM CATALOGO COMPLETO
4999 REM ***********************
5000 LET z$="": CAT #14,d
5010 CLS : POKE 23766,d: POKE 23768,15
5020 PRINT  INVERSE 1;"Titolo:";z$( TO 10)
5030 LET z$=z$(13 TO )
5040 IF LEN z$<10 THEN  GO TO 5330
5050 FOR i=1 TO 10
5060 POKE USR "a"+i-1,CODE z$(i)
5070 NEXT i
5080 CLOSE #15: LET z=USR open
5090 PRINT z$( TO 10);"  ";
5100 IF z=2 THEN  PRINT "File Dati": GO TO 5300
5110 LET a$="": FOR i=1 TO 9
5120 LET a$=a$+INKEY$#15: NEXT i
5130 LET z=CODE a$(1)
```

```
5140 GO TO 5150+z*40
5149 REM z=0 Programma
5150 PRINT "Programma   LINE ";CODE a$(8)+256*CODE a$(9)
5160 GO TO 5300
5189 REM z=1  Array Numerico
5190 PRINT "Array  ";CHR$ (CODE a$(6)-32);"()"
5200 GO TO 5300
5229 REM z=2  Array Stringa
5230 PRINT "Array  ";CHR$ (CODE a$(6)-96);"$()"
5240 GO TO 5300
5269 REM z=3  Code
5270 PRINT "Code  ";CODE a$(4)+256*CODE a$(5);
5280 PRINT ",";CODE a$(2)+256*CODE a$(3)
5300 CLOSE #15
5310 LET z$=z$(12 TO )
5320 GO TO 5040
5330 PRINT 'z$(2 TO LEN z$-1);"K byte liberi"
5340 RETURN
```

## Come generare file non di dati

Analogamente alla possibilità di leggere dalla cartuccia file non di dati, è anche possibile crearli da un programma BASIC. Non è necessario il codice macchina, ma soltanto un POKE. Poiché l'uso di questo programma è un po' complesso, non consiglio ai principianti di provarlo, in quanto qualunque errore può bloccare il computer. Occorre far seguire il comando OPEN # usato per creare un file da linee come

```
100 DEF FN p(p)=PEEK p+256*PEEK (p+1)
110 POKE FN p(s*2+23566+8)+FN p(23631)+66,4
```

dove s è il numero del canale logico usato nel comando OPEN. Si inganna così il sistema facendogli creare segmenti di file non di dati, quando qualunque dato sul canale logico viene scritto sulla cartuccia. Dopo il POKE, dovete eseguire il PRINT di nove caratteri sul canale logico scelto. Il primo identifica il tipo di file, i successivi otto i parametri del file, e sono gli stessi già menzionati riguardo ai file di lettura — le variabili di sistema HD_00 – HD_11 (vedi Appendice A). Ad esempio, il seguente programma salva su file tutte le variabili BASIC, come la routine presentata nel Capitolo 2, ma con una grande differenza: invece di eseguire il salvataggio sotto forma di due file CODE, salva le variabili sotto forma di file di programma, così che possano essere sfruttate in altri programmi. Questo metodo può esser più lento del precedente, ma è molto più flessibile.

**Routine SAVE VARIABILI**

```
3999 REM salva le variabili sotto forma di programmi
4000 OPEN #4;"m";1;"variabili"
4010 DEF FN p(p)=PEEK p+256*PEEK (p+1)
4020 LET d=FN p(4*2+23566+8)+FN p(23631)-1
4030 IF PEEK (d+4)<>CODE "M" THEN PRINT "Errore": STOP
4040 POKE d+67,4: REM inganna il sistema operativo
4045 FOR i=1 TO 2: NEXT i
4050 LET z=FN p(23641)-FN p(23627)-1
4055 LET z=FN p(23641)-FN p(23627)-1
4060 PRINT #4;CHR$ 0;: REM flag di programma
4070 PRINT #4;CHR$ (z-256*INT (z/256));CHR$ INT (z/256
);
4080 PRINT #4;CHR$ PEEK 23627;CHR$ PEEK 23628;: REM in
izio
4090 PRINT #4;CHR$ 0;CHR$ 0;: REM lunghezza programma
4100 PRINT #4;CHR$ 255;CHR$ 255;: REM  numero di linea
 per esecuzione automatica
4110 FOR i=FN p(23627) TO FN p(23627)+z-1
4120 PRINT #4;CHR$ PEEK i;
4130 NEXT i
4150 CLOSE #4
4160 RETURN
```

La linea 4000 crea il file di scrittura, quindi la linea 4040 esegue i necessari POKE. Le linee 4045 e 4050 possono sembrare superflue, ma sono entrambe vitali. La variabile z contiene il numero di byte nell'area variabili (linea 4055) e la linea 4060 mette il primo carattere per segnalare un file di programma. La linea 4070 scrive i due caratteri che definiscono la lunghezza del file, la linea 4080 i due che definiscono l'inizio dell'area variabili. La linea 4090 pone a 0 la lunghezza del programma e la linea 4100 seleziona 65535 come numero di linea di auto-esecuzione (cioè non esiste autostart). Il loop da 4110 a 4130 manda ogni byte delle variabili al file, prima che venga chiuso dalla linea 4150.

## Altri modi di leggere i dati

MOVE può essere più utile di INPUT # e INKEY$ # che sono i comandi più usati nella lettura dei file. Se la routine CANALE LOGICO 14-z$ è attiva, la linea

LET z$="": MOVE "m";1;"file" TO #14

leggerà in un'unica soluzione l'intero file di dati dalla cartuccia e lo tra-

sferirà nella variabile a stringa z$ (memoria permettendo). Questa variabile a stringa potrà quindi essere manipolata e modificata come si desidera, quindi scritta su un altro file (della cartuccia) con un singolo comando di PRINT:

OPEN #4;"m";1;"file2": PRINT #4;z$;: CLOSE #4

Questo metodo diminuisce l'usura delle cartucce, il che può essere importante trattando grandi file di dati da usare frequentemente, in quanto il drive lavorerà solo due volte, una per leggere ed una per scrivere.

## Routine STATO

La routine seguente, chiamata STATO, permette ad un programma di determinare se un particolare Microdrive sia o meno connesso, se contenga una cartuccia e se sia protetto contro la sovrascrittura. Prima di eseguire GO SUB 8550, la variabile stat dovrà essere inizializzata alla locazione di memoria desiderata.

**Routine STATO**

```
8545 REM *******************
8546 REM *      Stato      *
8547 REM *******************
8548 REM stat=indirizzo iniziale
8549 REM preferibilmente:64960 / 32190
8550 RESTORE 8630: LET c=0
8560 LET x2=INT ((stat+100)/256): LET x1=stat-256*x2+1
00
8570 FOR i=stat TO stat+128
8580 READ a: LET c=c+a
8590 POKE i,a
8600 NEXT i
8610 IF c<>14341+4*(x1+x2) THEN  PRINT "Errore di chec
ksum": STOP
8620 RETURN
8630 DATA 58,214,92,243,24,33,33,136
8640 DATA 19,43,125,180,32,251,33,136
8650 DATA 19,6,6,219,239,230,4,32
8660 DATA 4,16,248,24,84,43,124,181
8670 DATA 32,239,1,0,0,24,84,17
8680 DATA 0,1,237,68,198,9,79,6
8690 DATA 8,13,32,20,122,50,247,0
8700 DATA 62,238,211,239,205,x1,x2,62
```

```
8710 DATA 236,211,239,205,x1,x2,24,17
8720 DATA 62,239,211,239,123,211,247,205
8730 DATA x1,x2,62,237,211,239,205,x1
8740 DATA x2,16,214,122,211,247,62,238
8750 DATA 211,239,24,162,197,245,1,135
8760 DATA 0,11,120,177,32,251,241,193
8770 DATA 201,219,239,230,1,1,1,0
8780 DATA 32,1,3,197,175,207,33,193
8790 DATA 201
```

Il GO SUB si limita a porre in memoria il codice macchina senza eseguirlo. Prima dell'uso ponete nella cella 23766 il numero del drive e quindi usate alcune linee del tipo

```
2500 LET a=USR stat
2510 IF a=0 THEN PRINT "Microdrive non connesso"
2520 IF a=1 THEN PRINT "Cartuccia presente"
2530 IF a=2 THEN PRINT "Cartuccia protetta contro la sovrascrit-
     tura"
```

## Comandi di colore

Nel manuale dell'Interface 1, viene menzionato il fatto che i comandi di colore non possono funzionare usando canali fisici diversi da "K", "S" o "P". In effetti è un problema serio, di cui la persona desiderosa di lavorare con i file dovrebbe essere a conoscenza.
Il problema è il seguente: usando i canali dell'Interface 1 come output, i comandi di colore (ad esempio PAPER 3) non hanno apparentemente alcun effetto. Essi inviano dati ai file di scrittura. Per correggere questo, prima di predisporre colori permanenti, eseguite un comando di PRINT. Provate il seguente programma:

```
10 OPEN #4;"m";1;"coltest"
20 PRINT #4;"Prima linea"
30 FLASH 1: PAPER 4: CLS
40 PRINT #4; "Seconda linea"
50 CLOSE #4
60 MOVE "m";1;"coltest" TO #2
```

La linea 10 crea un file di scrittura "coltest" e la linea 20 gli invia una linea di dati. La linea 30 dovrebbe far lampeggiare in verde l'intero schermo, invece invia i codici di colore alla cartuccia. Le linee 40 e 50 man-

dano al file un'altra linea di dati e quindi chiudono il file. La linea 60, infine, visualizza sullo schermo l'intero file, rivelando l'indesiderata inserzione dei codici di colore. Per evitare questo inconveniente, aggiungete la linea

25 PRINT;

## Modifiche al programma "Tabella dei canali 1"

Le seguenti linee possono essere aggiunte al programma TABELLA DEI CANALI 1 per ottenere più dettagli circa i canali logici di Microdrive.

**Programma TABELLA DEI CANALI 2**

```
1630 IF F$="M" THEN  GO TO 2500
1840 IF FN P(D)<>8 AND FN P(D+2)<>8 THEN  RETURN
1850 PRINT "ROM ombra output :";FN P(D+5)
1860 PRINT "ROM ombra input  :";FN P(D+7)
2499 REM  Canale M
2500 PRINT  INVERSE 1;"MICRODRIVE"
2510 GO SUB 1800
2520 PRINT "Numero Drive      :";PEEK (D+25)
2530 LET M=FN P(D+26)
2540 PRINT "Mappa del nastro:"
2550 FOR I=0 TO 31: FOR J=1 TO 6
2560 POKE 16384+2048+2*32+J*256+31-I,PEEK (M+I)
2570 NEXT J: NEXT I
2575 PLOT 0,95: DRAW 255,0: PLOT 0,88: DRAW 255,0
2580 PRINT '"Area Map          :";M;"-";M+31
2590 PRINT "Nome Cartuccia    :";
2600 FOR I=D+44 TO D+53
2610 PRINT CHR$ PEEK I;
2620 NEXT I: PRINT
2630 PRINT "Nome del File     :";
2640 FOR I=D+14 TO D+23
2650 PRINT CHR$ PEEK I;
2660 NEXT I: PRINT '"Spazio libero     :";
2670 LET F=0
2680 FOR I=0 TO 255
2690 LET F=F+NOT POINT (I,90)
2700 NEXT I
2710 PRINT F/2;"Kbyte"
2720 GO TO 1500
```

Questo programma visualizzerà anche il numero del Microdrive, il numero ed il nome della cartuccia ed il nome del file, nonché lo spazio disponibile sulla cartuccia. Viene inoltre visualizzata una "mappa" del nastro, una specie di codice a barre, che presenta graficamente le sezioni del nastro sfruttate. Osservate l'output generato dal programma con una cartuccia appena formattata. Le barrette nere sulla mappa del nastro segnalano aree che sono usate o non esistenti. La grossa area scura sulla sinistra, indica una sezione del nastro che semplicemente non esiste; il sistema può infatti leggere nastri più lunghi. La mappa del nastro di questa cartuccia contiene più barrette verticali, che mostrano dove sono fisicamente immagazzinati i programmi ed i dati.

**Output del programma TABELLA DEI CANALI 2**

```
           CANALE LOGICO 10

Identificatore di canale:M
MICRODRIVE
Routine Output      :8
Routine Input       :8
ROM ombra output    :4568
ROM ombra input     :4386
Numero Drive        :1
Mappa del nastro:

Area Map            :23792-23823
Nome Cartuccia      :pinin1
Nome del File       :Teresa
Spazio libero       :18.5Kbyte
```

La prima aggiunta importante al programma è costituita dalle linee 1840-1860. Queste controllano se il canale appartenga all'Interface; in questo caso vengono visualizzate le locazioni delle routine di input e di output della ROM ombra. Queste linee supplementari sono richieste anche dalle aggiunte al programma per la RS232 e la rete locale. Le linee 2500-2720 visualizzano vari dettagli sul canale logico. La mappa del nastro viene visualizzata tramite dei POKE direttamente nella memoria video, eseguiti nelle linee 2530-2575. Lo spazio libero viene calcolato contando il numero di barrette bianche nella mappa visualizzata sullo schermo (2670-2710).

## Punteggio massimo

Per dimostrare l'applicazione dei file di dati, presentiamo due routine che servono ad aggiungere il "punteggio massimo" ad un gioco, salvando la tabella in un file. Si tratta di due subroutine principali: quella di linea

9000 legge la tabella e va usata prima dell'inizio del gioco; quella di linea 9100 aggiorna la tabella se è stato raggiunto un nuovo punteggio massimo.

**Routine PUNTEGGIO MASSIMO**

```
8999 REM Legge i dati
9000 LET ns=5: DIM s(ns): DIM s$(ns,10)
9010 CLOSE #4: OPEN #4;"m";1;"hiscore"
9020 FOR i=1 TO ns
9030 INPUT #4;s(i); LINE s$(i)
9040 NEXT i: CLOSE #4
9050 GO TO 9300
9098
9099 REM Se viene raggiunto un nuovo punteggio massimo
, riscrive il file
9100 LET ns=5: IF sc<s(ns) THEN  RETURN
9110 PRINT ' FLASH 1;"  Un nuovo punteggio massimo!!!
"
9120 INPUT "Scrivi il tuo nome                (max 10 le
ttere)  "; LINE a$
9130 FOR i=1 TO ns
9140 IF sc<=s(i) THEN  NEXT i
9150 ERASE "m";1;"hiscore"
9155 OPEN #4;"m";1;"hiscore"
9160 FOR j=1 TO ns
9170 IF j<i THEN  PRINT #4;s(j)'s$(j)
9180 IF j=i THEN  PRINT #4;sc'a$
9190 IF j>i THEN  PRINT #4;s(j-1)'s$(j-1)
9200 NEXT j
9210 CLOSE #4
9220 RETURN
9299 REM Visualizza la tabella dei punteggi
9300 CLS
9310 PRINT  FLASH 1; INK 7; PAPER 0;TAB 6;"PUNTEGGI MA
SSIMI";TAB 31;" "
9320 FOR i=1 TO ns
9330 PRINT  PAPER i; INK 9,,TAB 6;i;" ";s$(i);TAB 20;s
(i),,,
9340 NEXT i
9350 RETURN
9988
9989 REM Predispone il file vuoto
9990 OPEN #4;"m";1;"hiscore"
9992 FOR i=1 TO 5
9994 PRINT #4;0'" "
9996 NEXT i: CLOSE #4
```

La tabella viene memorizzata sotto forma di 10 elementi, nell'ordine: primo punteggio, primo nome, secondo punteggio, secondo nome, ecc. Le linee da 9900 in poi, vanno eseguite una sola volta, perché inizializzano il file senza dati. Per leggere la tabella, vengono usate le linee 9000-9050; la linea 9000 predispone il numero di elementi della tabèlla (in ns), l'array s( ) per i punteggi ed s$( ) per i nomi. La linea 9010 apre il file per la lettura e le linee 9020-9040 leggono i punteggi e i nomi e li caricano nell'array. Il file viene poi chiuso e la tabella visualizzata nella linea 9300. Finito il gioco, occorrerà porre il punteggio raggiunto nella variabile sc ed eseguire un GO SUB 9100. La linea 9100 controlla se il punteggio sia inferiore a quello più basso presente nella tabella. Se è così, viene eseguito un RETURN, non essendosi il giocatore qualificato per essere memorizzato in tabella. In caso contrario, il giocatore introdurrà il suo nome (linea 9120) e la sua nuova posizione nella tabella sarà calcolata nelle linee 9130-9140 e posta in i. La linea 9150 cancella il vecchio file e la linea 9155 crea un nuovo file di scrittura. Il loop da 9160 a 9200 scrive nel file la nuova tabella, inserendo il nuovo punteggio ed il nome e slittando in basso di una posizione ogni dato al di sotto. Infine la linea 9210 chiude il file.

Capitolo

# Il programma Unifile 4

Il programma presentato in questo capitolo è un database che sfrutta i Microdrive e, opzionalmente, una stampante (tanto ZX che RS232). Esso è un'elaborazione, autorizzata dall'autore, del programma Unifile 1 di David Lawrence, pubblicato nel libro *The Working Spectrum.*
Il programma è stato concepito per lo Spectrum 48K, ma in seguito saranno dati dettagli per ottenerne una versione ridotta per i 16K. Il programma non è stato rinumerato, quindi le linee prelevate dal programma originale sono invariate.

**Programma UNIFILE**

```
1000 PAPER 7: CLS : BORDER 7: INK 6: PAPER 0: PRINT  P
APER 2;"   UNIFILE ";F$;TAB 31;" "
1005 LET all=0
1010 PRINT '"FUNZIONI DISPONIBILI:"
1020 PRINT '"      1)CREAZIONE DI UN FILE"
1030 PRINT '"      2)IMMISSIONE DATI"
1040 PRINT '"      3)RICERCA/VISUALIZZAZIONE/        MO
DIFICA"
1050 PRINT '"      4)STOP"
1055 PRINT '"      5)SAVE/LOAD/CAT"
1060 PRINT ''"SCEGLI."
1070 PAUSE 0: LET Z$=INKEY$
1080 CLS
1090 IF Z$="1" THEN  GO SUB 1210
1100 IF Z$="2" THEN  GO SUB 1440
1110 IF Z$="3" THEN  GO SUB 2180
```

```
1120 IF Z$="4" THEN  GO SUB 1150
1125 IF Z$="5" THEN  GO TO 3500
1130 CLS
1140 GO TO 1000
1150 PRINT AT 10,5; INK 7; PAPER 2;"FINE DELLE OPERAZI
ONI"
1160 BEEP 2,2
1180 INPUT "Hai introdotto nuovi dati che   desideri s
alvare? (S/N)";Q$: IF Q$="N" THEN  STOP
1190 SAVE "UNIFILE": PRINT '"Riavvolgi il nastro,quind
i premiun tasto per iniziare il VERIFY": PAUSE 0: VERI
FY "UNIFILE": STOP
1200 REM ***********************
1210 REM STRUTTURA RECORD
1220 REM ***********************
1230 PRINT  PAPER 2;"     STRUTTURA DEL FILE       "
1235 GO SUB 4200
1240 PRINT '"QUANTI CAMPI PER RECORD?"
1250 INPUT X
1260 CLS
1270 DIM A$(X,20)
1280 PRINT  PAPER 2;"    NOMI ATTRIBUITI AI CAMPI    "
1290 FOR I=1 TO X
1300 PRINT "CAMPO ";I;":";
1310 GO SUB 2780
1320 PRINT Q$(2 TO )
1330 LET A$(I)=Q$
1340 NEXT I
1350 DIM B$(28000)
1360 LET B$(1 TO 4)=CHR$ 2+CHR$ 0+CHR$ 2+CHR$ 255
1370 DEF FN A()=256*CODE Y$(2*S-1)+CODE Y$(2*S)
1380 DEF FN A$()=B$(C TO C+CODE B$(C)-1)
1390 LET P=5
1400 LET Y$=CHR$ 0+CHR$ 1+CHR$ 0+CHR$ 3
1410 LET N=2
1420 RETURN
1430 REM ***********************
1440 REM IMMISSIONE DATI
1450 REM ***********************
1460 LET R$=""
1470 PRINT  PAPER 2;"          CAMPI               "
1480 PRINT '"COMANDI DISPONIBILI:"
1490 PRINT '">INTRODUCI IL DATO"''">""ZZZ""PER FINIRE"
1500 PRINT "********************************"
1510 PRINT "MISURA FILE:";P-1;"/";LEN B$
1520 FOR I=1 TO X
1530 GO SUB 2810
1540 GO SUB 2780
1580 PRINT Q$(2 TO )
1590 IF Q$(2 TO )="ZZZ" THEN  RETURN
```

```
1600 LET R$=R$+Q$
1610 NEXT I
1620 CLS
1630 GO SUB 1660
1640 GO TO 1440
1650 REM ***********************
1660 REM PONE I DATI NEL FILE
1670 REM ***********************
1680 IF P+LEN R$-1<LEN B$ THEN  GO TO 1730
1690 PRINT AT 14,10;"IL FILE E' PIENO"
1700 PRINT ''"   Premi un tasto per continuare"
1710 PAUSE 0
1720 RETURN
1730 LET POWER=INT (LN (N-1)/LN 2)
1740 LET S=2^POWER
1750 LET T$=R$(2 TO CODE R$(1))
1760 FOR K=POWER-1 TO 0 STEP -1
1770 LET C=FN A()
1780 LET U$=FN A$()(2 TO )
1790 LET S=S+(2^K)*(T$>U$)-(2^K)*(T$<U$)
1810 IF S>N-1 THEN  LET S=N-1
1820 IF S<2 THEN  LET S=2
1830 NEXT K
1840 LET C=FN A()
1850 LET U$=FN A$()(2 TO )
1860 IF T$<U$ THEN  LET S=S-1
1870 LET B$(P TO P+LEN R$-1)=R$
1880 LET N=N+1
1890 LET Y$=Y$(1 TO 2*S)+CHR$ INT (P/256)+CHR$ (P-256*
INT (P/256))+Y$(2*(S+1)-1 TO )
1900 LET P=P+LEN R$
1910 RETURN
1920 REM ***********************
1930 REM MODIFICA DATI
1940 REM ***********************
1950 LET S=S-1
1960 LET C=FN A()
1970 LET R$=""
1980 PRINT "RECORD ";S-1;":-"
1990 FOR I=1 TO X
2000 GO SUB 2810
2010 GO SUB 2830
2020 PRINT AT 17,0; PAPER 2;"        VARIAZIONI
     "
2030 PRINT "COMANDI DISPONIBILI:"
2040 PRINT ">""ENTER""LASCIA IL DATO INVARIATO"'">""ZZ
Z"" CANCELLA TUTTO IL RECORD"'">INTRODUCI IL NUOVO DAT
O"
2050 GO SUB 2780
2060 IF LEN Q$=1 THEN  LET R$=R$+B$(C TO C+CODE B$(C)-
1)
```

```
2070 LET C=C+CODE B$(C)
2080 CLS
2090 IF LEN Q$=1 THEN  GO TO 2120
2100 IF Q$(2 TO )="ZZZ" THEN  GO TO 2130
2110 LET R$=R$+Q$
2120 NEXT I
2130 GO SUB 3130
2140 IF Q$(2 TO )="ZZZ" THEN  RETURN
2150 GO SUB 1660
2160 RETURN
2170 REM ***********************
2180 REM RICERCA
2190 REM ***********************
2200 LET S=2
2205 LET ST=2
2210 PRINT  PAPER 2;"            RICERCA           "
2220 PRINT ''"COMANDI DISPONIBILI:"
2230 PRINT ">INSERISCI IL DATO DA RICERCARE"'">PRECEDU
TO DA ""SSS"" PER"'" RICERCA SPECIALE "'">PRECEDUTO DA
 ""III"" PER CERCARE  IL PRIMO CARATTERE DEL RECORD"'"
>""ENTER"" PER IL PRIMO DATO SUL   FILE"
2235 PRINT ">""PPP"" PER USCITA SU ";"STAMPANTE" AND S
T=2;"VIDEO" AND ST=3
2240 PRINT "********************************"
2250 PRINT ''" INSERISCI IL DATO DA CERCARE "';
2260 GO SUB 2780
2265 IF Q$=CHR$ 4+"PPP" THEN  LET ST=5-ST: CLS : GO TO
 2210
2270 PRINT Q$(2 TO )
2280 LET S$=Q$
2290 IF LEN S$=1 THEN  GO TO 2510
2300 LET C=FN A()
2310 IF LEN S$<5  THEN  GO TO 2430
2320 IF S$(2 TO 4)<>"III" THEN  GO TO 2390
2330 FOR I=S TO N
2340 LET S=I
2350 LET C=FN A()
2360 IF B$(C+1)=S$(5) THEN  GO TO 2510
2370 NEXT I
2380 RETURN
2390 IF S$(2 TO 4)<>"SSS" THEN  GO TO 2430
2400 GO SUB 2920
2410 IF C4=1 THEN  GO TO 2510
2420 RETURN
2430 FOR I=1 TO X
2440 IF FN A$()=S$ THEN  GO TO 2510
2450 IF FN A$()=CHR$ 2+CHR$ 255 THEN  RETURN
2460 LET C=C+CODE B$(C)
2470 NEXT I
2480 LET S=S+1
```

```
2490 LET C=FN A()
2500 GO TO 2430
2510 LET C=FN A()
2520 LET C4=0
2530 IF FN A$()=CHR$ 2+CHR$ 255 THEN  RETURN
2540 CLS
2545 IF ST<>2 THEN  GO TO 4300
2550 PRINT "RECORD ";S-1;":-"
2560 GO SUB 2850
2570 LET S=S+1
2580 PRINT AT 15,0; PAPER 2;"          RICERCA
     "
2590 PRINT "COMANDI DISPONIBILI:"
2600 PRINT ">""ENTER"" PER VEDERE IL PROSSIMO  DATO"'"
>""ZZZ"" PER FINIRE "'">""AAA"" PER CORREGGERE "'">""C
CC"" PER CONTINUARE LA RICERCA"
2610 INPUT P$
2620 CLS
2630 IF P$="CCC" THEN  GO TO 2300
2640 IF P$="" THEN  GO TO 2510
2650 IF P$<>"AAA" THEN  GO TO 2710
2660 LET C=FN A()
2670 CLS
2680 GO SUB 1930
2710 IF P$="ZZZ" THEN  RETURN
2720 IF P$="AAA" THEN  RETURN
2730 CLS
2740 GO TO 2260
2750 REM ***********************
2760 REM SUBROUTINE FUNZIONALI
2770 REM ***********************
2780 INPUT q$
2790 LET Q$=CHR$ (LEN Q$+1)+Q$
2800 RETURN
2810 PRINT A$(I,2 TO CODE A$(I,1));":";
2820 RETURN
2830 PRINT FN A$()(2 TO )
2840 RETURN
2850 FOR I=1 TO X
2860 GO SUB 2810
2870 GO SUB 2830
2880 LET C=C+CODE B$(C)
2890 NEXT I
2900 RETURN
2910 REM ***********************
2920 REM RICERCA SPECIALE
2930 REM ***********************
2940 LET C4=0
2950 FOR H=S TO N-1
2960 LET S=H
```

```
2970 LET C=FN A()
2980 LET C1=C
2990 FOR I=1 TO X
3000 LET C1=C1+CODE B$(C1)
3010 NEXT I
3020 FOR J=C+1 TO C1-LEN S$+5
3030 IF B$(J TO J+LEN S$-5)<>S$(5 TO ) THEN  GO TO 3060
3040 LET C4=1
3050 RETURN
3060 NEXT J
3070 NEXT H
3080 LET C4=0
3090 RETURN
3100 REM ***********************
3110 REM FILE TELESCOPICO
3120 REM ***********************
3130 LET C=FN A()
3140 LET SHIFT=1000
3150 LET C1=C
3160 LET C3=C
3170 FOR I=1 TO X
3180 LET C1=C1+CODE B$(C1)
3190 NEXT I
3200 LET C2=C1-C
3210 FOR I=C1 TO LEN B$-1 STEP SHIFT
3220 IF LEN B$-I+1<SHIFT THEN  LET SHIFT=LEN B$-I+1
3230 LET S$=B$(I TO I+SHIFT-1)
3240 LET B$(C TO C+SHIFT-1)=S$
3250 LET C=C+SHIFT
3260 NEXT I
3270 LET Y$=Y$(1 TO 2*(S-1))+Y$(2*(S+1)-1 TO )
3280 FOR I=1 TO N-1
3290 LET S=I
3300 LET C=FN A()
3310 IF C<=C3 THEN  GO TO 3350
3320 LET C=C-C2
3330 LET Y$(2*I-1)=CHR$ INT (C/256)
3340 LET Y$(2*I)=CHR$ (C-256*INT (C/256))
3350 NEXT I
3360 LET P=P-C2
3370 LET N=N-1
3380 RETURN
3390 FOR I=1 TO 20: PRINT CODE Y$(I): NEXT I
3500 CLS
3510 PRINT "UNIFILE - OPERAZIONI MICRODRIVE"
3520 PRINT '"  1) SAVE "; INVERSE 1;F$
3530 PRINT '"  2) LOAD nuovi dati"
3540 PRINT '"  3) CATaloga la cartuccia"
3545 PRINT '"  4) RITORNA al menu principale"
3550 PAUSE 0: LET Z$=INKEY$
```

```
3560 IF Z$="1" THEN  GO SUB 3900
3570 IF Z$="2" THEN  GO TO 3800
3580 IF Z$="3" THEN  GO SUB 3700
3590 IF Z$="4" THEN  GO TO 1000
3600 GO TO 3500
3670 REM ********************
3680 REM ROUTINE CATALOGO
3690 REM ********************
3700 CLS : PRINT "CATALOGO CARTUCCIA:"
3710 INPUT "NUMERO DRIVE? (0 per finire)";D
3720 IF D=0 OR D>8 THEN  RETURN
3730 CAT D
3740 PRINT #0;AT 0,0; FLASH 1;" Premi un tasto per con
tinuare  ";
3750 PAUSE 0
3760 RETURN
3770 REM *******************
3780 REM ROUTINE LOAD
3790 REM *******************
3800 PRINT '' FLASH 1;"ATTENZIONE:CARICANDO NUOVI DATI
,SI CANCELLANO QUELLI PRESENTI   "
3805 PRINT '"Premi C per continuare,qualunquealtro tas
to per uscire."
3810 PAUSE 0: LET Z$=INKEY$: IF Z$<>"C" AND Z$<>"c" TH
EN  GO TO 3500
3820 GO SUB 4200
3830 INPUT "NUMERO DRIVE (1-8)? ";D
3840 IF D<0 OR D>8 THEN  GO TO 3830
3850 LOAD *"M";D;F$+CHR$ 128
3860 GO TO 1000
3870 REM *******************
3880 REM ROUTINE SAVE
3890 REM *******************
3900 PRINT ''"(NOME PRESENTE=";F$;")"
3905 GO SUB 4200
3910 INPUT "NUMERO DRIVE? ";D
3920 IF D<0 OR D>8 THEN  RETURN
3930 CLS
3940 PRINT "   UNIFILE - ";F$
3950 PRINT AT 15,0;"Vuoi che il file ";F$;" sul"'"driv
e ";D;" sia cancellato? (S/N)"
3960 INPUT  LINE z$
3965 PRINT AT 15,0;,,,,
3970 IF Z$<>"S" AND Z$<>"s" THEN  GO TO 4000
3980 ERASE "M";D;F$+CHR$ 128
3990 PRINT '"FILE CANCELLATO"
4000 SAVE *"M";D;F$+CHR$ 128 LINE 1000
4010 PRINT "DATI SALVATI - VERIFICA IN CORSO"
4020 VERIFY *"M";D;F$+CHR$ 128 LINE 1000
4030 RETURN
```

```
4170 REM ***********************
4180 REM INTRODUZIONE NOME FILE
4190 REM ***********************
4200 INPUT "NOME FILE? "; LINE F$
4210 IF LEN F$<10 AND LEN F$>0 THEN  RETURN
4220 INPUT "NOME FILE? (MAX 9 CARATTERI)"'; LINE F$
4230 GO TO 4210
4270 REM ***********************
4280 REM USCITA SU STAMPANTE
4290 REM ***********************
4300 LPRINT "RECORD ";S-1;":-"
4310 FOR I=1 TO X
4320 LPRINT A$(I,2 TO CODE A$(I,1));":";
4330 LPRINT FN A$()(2 TO )
4340 LET C=C+CODE B$(C)
4350 NEXT I
4360 GO TO 2570
```

Introdotto il programma, digitate:

CLEAR
LET F$=""

e salvate il programma in modo che la sua esecuzione automatica inizi dalla linea 1000. Per eseguirlo, usate GO TO 1 e non digitate mai RUN, che cancellerebbe i dati.
Ogni file di dati può contenere fino a 28000 caratteri, che compongono un certo numero di argomenti, definito all'inizializzazione del file. Consideriamo, ad esempio, un indirizzario, dotato di tre campi per record: Nome, Indirizzo, Numero di telefono.
L'opzione 1 predispone un nuovo file, di cui viene richiesto il nome.
L'opzione 2 vi permette di inserire dati nel file.
L'opzione 3 è la più utile: essa vi permette di ricercare dati attraverso l'intero file, visualizzando gli opportuni record sullo schermo o stampandoli. L'opzione di "ricerca speciale" scandisce ciascuna parte di ogni record per l'occorrenza di una determinata stringa e non è molto veloce. Digitando, ad esempio, SSS-MILANO, e supponendo di operare sull'indirizzario, sarà visualizzato il nome di qualunque persona abiti a MILANO, o il cui cognome sia appunto MILANO. Un'opzione più veloce, che si limita a controllare il primo carattere di ogni record, è la ricerca normale.
L'opzione 4 permette di salvare programma e dati su cassetta per archivio e copie di sicurezza.
L'opzione 5 permette di salvare i dati correnti, o di caricare altri dati, o di ottenere il catalogo di una cartuccia. I file vengono salvati sulle cartucce con un CHR$ 128 aggiunto in coda, per distinguerli da un normale

file di programma. Il carattere aggiuntivo non viene notato nel catalogo, perché viene visualizzato come uno spazio.
Non cercherò di spiegare il funzionamento di questo programma, che è ben chiarito da David Lawrence nel suo libro. Illustrerò tuttavia la subroutine di stampa, in modo che l'utente possa modificarla per soddisfare le proprie necessità. Questa routine va dalla linea 4300 alla linea 4350. S-1 è la posizione del record nel file ed X è il numero di campi per record. La linea 4320 stampa il titolo di ogni record e la linea 4330 i dati. La linea 4340 incrementa un puntatore. Per esempio, supponendo sempre di operare sull'indirizzario e desiderando stampare i vari record sotto forma di etichette da applicare sulle buste, potreste usare:

```
4300 LPRINT "Ref n.";S-1
4310 LPRINT FN A$( )(2 TO): REM Nome
4320 LET C=C+CODE B$(C)
4330 LPRINT FN A$( )(2 TO): REM Indirizzo
4340 LPRINT
4350 LET C=C+CODE B$(C)
4360 LPRINT "Telefono";FN A$( )(2 TO): REM Numero di telefono
4370 LET C=C+CODE B$(C)
4380 GO TO 2570
```

È possibile aggiungere ulteriori caratteristiche e capacità al programma. Sarebbe possibile, ad esempio, ottenere in seguito al CAT la visualizzazione dei soli programmi di UNIFILE, usando la routine CANALE LOGICO 14-z$ ed esaminando Z$ alla ricerca di CHR$ 128. Un'altra miglioria potrebbe consistere nella conversione delle linee 3130-3380 in codice macchina, la qual cosa aumenterebbe enormemente la velocità delle cancellazioni e delle variazioni.

## Versione 16K

È possibile usare una versione ridotta di UNIFILE su uno Spectrum 16K, eseguendo le seguenti modifiche: cancellate tutte le REM e le linee 3540, 3580 e 3700-3760 e modificate le linee

```
1350 DIM B$(1000)
3140 LET SHIFT=100
3375 LET S$=""
```

Capitolo

# 5 Protezione dei programmi

Se avete scritto un buon programma e desiderate commercializzarlo, vorrete evitare, ovviamente, che chiunque sia in grado di ottenerne copie da distribuire agli amici. Anche se nessun programma è incopiabile al 100%, esistono varie semplici tecniche che rendono l'operazione di copiatura quantomeno difficoltosa: si tratta di un paio di caratteristiche proprie dello Spectrum e di alcuni altri metodi più raffinati. È bene chiarire subito che nessun metodo è imbattibile. Un buon programmatore in codice macchina sarà sempre in grado di aggirare qualunque metodo di protezione di un programma, purché abbia sufficiente tempo e conoscenza.

## Autoesecuzione

Il metodo più semplice per rendere difficoltosa la copiatura dei programmi consiste nel predisporli per l'autoesecuzione usando la funzione SAVE *...LINE. Un programma così concepito non può essere fuso (MERGE) con altri e partirà sempre automaticamente in esecuzione una volta caricato. Se poi chiamate il programma "run" (tutto minuscolo), all'accensione o dopo un NEW, la digitazione della parola RUN caricherà il programma e ne inizierà l'esecuzione (se era stato salvato con l'opzione LINE).
È molto utile che i nomi dei programmi inizino con CHR$ 0, che ne impedisce la visualizzazione nel catalogo della cartuccia. Se fate sì che il programma "run" carichi quello "invisibile", il potenziale pirata del software non saprà mai il nome del programma principale. Un altro sistema

consiste nell'assegnare al programma un nome formato esclusivamente da spazi, la cui presenza nel catalogo sfuggirà alla gran parte degli utenti.

## POKE di autodistruzione

Occorre impedire all'utente di interrompere l'esecuzione del programma o il suo caricamento. Nel primo caso questo gli permetterebbe di vedere il nome del file invisibile, semplicemente esaminando il listato. Un modo di protezione non particolarmente ingegnoso, ma che funziona bene, consiste nell'usare come prima linea

```
POKE 23659,0
```

Questo informa lo Spectrum che non sono presenti linee nella parte inferiore dello schermo. Quindi, in caso di errore, (come BREAK INTO PROGRAM) l'intero sistema si bloccherà, poiché non esiste spazio dove visualizzare il messaggio di errore. Sarà pertanto necessario spegnere il computer, la qual cosa chiaramente impedirà di esaminare il listato o di copiarlo. Attenzione all'uso di questo particolare POKE: un comando CLS, o ogni tentativo di visualizzare sulle due linee inferiori (ad esempio usando INPUT) bloccherà il computer. POKE è in effetti veramente utile solo per i programmi in codice macchina e solo per il loro caricamento; per i programmi BASIC non è consigliabile, in quanto è troppo limitante. Un metodo alternativo consiste nell'uso della linea

```
LET p=PEEK 23613+256*PEEK 23614: POKE p,0: POKE p+1,0
```

che bloccherà il sistema all'occorrenza di qualunque errore, permettendo però la visualizzazione nella parte inferiore dello schermo.

## ON ERROR GO TO

Molti computer hanno la funzione ON ERROR GO TO che, in caso di errore, genera un salto ad una certa linea. La mancanza di una siffatta routine nello Spectrum è superata con il seguente programma in codice macchina:

**Programma ON ERROR**

```
8794 REM ********************
8795 REM *  ON ERROR GO TO  *
8796 REM ********************
8797 REM err=indirizzo iniziale,line=linea cui cedere
il controllo in caso di errore
8798 REM preferibilmente:64780 / 32010
8800 RESTORE 8900: LET c=0
8810 FOR i=err TO err+170
8820 READ a: LET c=c+a
8830 IF a<256 THEN  POKE i,a
8840 NEXT i
8850 IF c<>17439 THEN  PRINT "Errore di checksum": STOP
8860 POKE err+149,line-256*INT (line/256)
8870 POKE err+150,INT (line/256)
8875 RETURN
8880 RANDOMIZE USR err
8885 POKE 23734,0: RETURN
8890 RANDOMIZE USR err
8895 POKE 23734,4: RETURN
8900 DATA 33,17,0,9,235,42,61,92
8905 DATA 115,35,114,207,49,1,0,0
8910 DATA 201,34,201,92,42,61,92,17
8915 DATA 3,19,213,205,176,22,253,203
8920 DATA 55,174,205,110,13,33,185,23
8925 DATA 34,237,92,58,58,92,50,211
8930 DATA 92,245,207,50,33,129,0,237
8935 DATA 91,201,92,241,167,237,82,32
8940 DATA 20,253,203,124,70,32,14,254
8945 DATA 7,40,10,62,100,50,211,92
8950 DATA 62,63,215,24,21,60,71,254
8955 DATA 10,56,2,198,7,205,239,21
8960 DATA 62,32,215,120,17,145,19,205
8965 DATA 10,12,175,17,54,21,205,10
8970 DATA 12,237,75,69,92,237,67,201
8975 DATA 92,205,27,26,62,58,215,253
8980 DATA 78,13,6,0,205,27,26,33
8985 DATA 59,92,203,174,251,203,110,40
8990 DATA 252,33,66,92,17,260,270,115
8994 DATA 35,114,35,54,1,253,54,0
8995 DATA 255,253,54,124,0,205,110,13
8996 DATA 195,125,27
```

Questo programma è lungo 171 byte ed è indipendente dalla posizione. Prima di digitare il GO SUB 8800 per ottenere la rilocazione, la variabile err dovrà essere predisposta all'indirizzo iniziale desiderato e la variabile line al numero di linea a cui si desidera cedere il controllo dopo l'errore.

Se avviene un errore, l'opportuno messaggio verrà visualizzato al posto consueto. Si attenderà quindi la pressione di un tasto prima di cancellare il messaggio e cedere il controllo alla linea desiderata. La caratteristica viene disattivata dopo ogni errore. Il numero di linea dove si è verificato l'errore può essere ottenuto da

```
PEEK 23753+256*PEEK 23754
```

e il numero di errore con

```
PEEK 23763
```

Durante le operazioni con l'Interface 1, dovrete attivare la routine tramite GO SUB 8890. Nel caso avvenga un errore di interfaccia (cioè privo di codice), non sarà prodotto alcun messaggio e verrà invece visualizzato un punto interrogativo; la locazione 23763 conterrà 100.
Dopo ogni comando di interfaccia, la routine viene disabilitata. Se desiderate disabilitare la routine, eseguite:

```
LET p=PEEK 23613+256*PEEK 23614: POKE p,3: POKE
p+1,19: POKE 23734,0
```

Questa routine non è compatibile con la routine ON EOF GO TO presentata nel capitolo precedente. Per controllare se è avvenuto un errore di EOF, basterà esaminare il contenuto della variabile 23763, che sarà 7 in caso di EOF.
Durante il debugging, prestate particolare attenzione ad evitare loop infiniti. Se 1000 era il numero di linea cui cedere il controllo dopo un errore, ed era

```
1000 GO SUB 8880
```

non sarete mai in grado di uscire dal loop; includete quindi da qualche parte nel vostro programma un'opzione per permettere la disabilitazione della routine. Se desiderate includere questa routine in un programma destinato alla vendita aggiungete la seguente linea:

```
8855 POKE err+23,94: POKE err+24,35: POKE err+25,86
```

Questi POKE alterano la routine in modo che non venga cancellata in seguito ad un errore.
Come abbiamo visto, è possibile sfruttare tanto il codice macchina che alcuni POKE per proteggere il programma, ma è vero anche il contrario: il codice macchina può essere anche usato per infrangere qualunque pro-

tezione. Il programmatore che desideri aggirare le protezioni, prima dovrà conoscere quale è il metodo usato. Ma è abbastanza facile per un buon programmatore in codice macchina ottenere un catalogo di file invisibili, o fondere programmi predisposti per l'autoesecuzione. È uno spiacevole effetto collaterale dell'abbondante disponibilità di dati tecnici su tutti i prodotti Sinclair.

Capitolo

# Uso dell'interfaccia RS232

# 6

Una caratteristica dell'Interface 1 è la possibilità di trasmettere e ricevere dati per mezzo della RS232. RS232 è uno standard pressoché universale di trasmissione seriale: esiste un gran numero di periferiche e di computer in grado di comunicare via RS232. Per quanto riguarda lo Spectrum, l'applicazione più comune consiste nel pilotaggio di stampanti per ottenere listati e tabulati chiari e leggibili.
La velocità di trasmissione o ricezione dati viene chiamata *baud rate* ed è pari ad approssimativamente 10 volte il numero di byte trasmessi in un secondo. I baud rate standard sono nove, tutti supportati dallo Spectrum: 50, 110, 300, 600, 1200, 2400, 4800, 9600, e 19200. Le telescriventi normalmente usano le basse velocità di 110 o 300 baud, mentre i terminali veloci sfruttano le velocità da 2400 baud in su.
I vari dispositivi, oltre a differenti velocità, possono richiedere anche differenti protocolli di trasmissione; lo Spectrum usa il seguente:

— nessun bit di parità
— otto bit di dati
— un bit di stop

Per sfruttare la RS232 sullo Spectrum, vengono usati altri due identificatori di canale: "t" e "b". Questi canali logici si differenziano nella maniera in cui il singolo carattere viene trattato.

## Canale di testo "t"

Il canale di testo è il più idoneo al pilotaggio delle stampanti, poiché esegue determinate operazioni su ogni carattere. Facendo riferimento al set di caratteri presentato nel manuale dello Spectrum, il canale "t" tratta i codici di carattere nel modo seguente:

| | |
|---|---|
| 0−12 | (codici di controllo) sono ignorati |
| 13 | (a capo) genera un ritorno carrello e un salto di linea |
| 14−31 | (codici di controllo) sono ignorati |
| 32−127 | (codici ASCII) vengono trasmessi senza alcuna modifica |
| 128−164 | (codici grafici) vengono trasmessi come "?" |
| 165−255 | (codici delle parole chiave del BASIC, in inglese *tokens*) vengono decodificati in singoli caratteri. |

Questa conversione dei caratteri è ideale per stampare listati e tabulati, ma molte stampanti usano i codici inferiori a 32 per ottenere speciali funzioni; ad esempio, le *Epson* usano CHR$14 per ottenere il modo espanso, ma questo carattere non può essere trasmesso sul canale "t". Inoltre, l'utilissima funzione TAB viene ignorata, rendendo quindi più difficile una buona impaginazione. È però possibile usare il canale "b".

## Canale binario "b"

Il canale binario è molto utile nella comunicazione con dispositivi diversi dalla stampante, poiché non esegue alcuna conversione di carattere. Ogni carattere viene trasmesso non modificato; ciò rende il canale ideale alla trasmissione di dati e programmi. Il suo uso è inoltre necessario per inviare i caratteri di controllo alle stampanti, poiché, come abbiamo già visto, questi non possono essere trasmessi attraverso il canale "t".
Per mettere lo Spectrum in grado di trasmettere o ricevere dati attraverso la RS232, è necessario adattare la velocità a quella della periferica. Ciò è ottenuto tramite il comando FORMAT "b"; seguito dalla velocità. Volendo usare il baud rate 110, digitate

```
FORMAT "b";110
```

(Il comando FORMAT "t";110 ha lo stesso effetto). Se introducete una velocità non standard, non verrà visualizzato alcun messaggio di errore e lo Spectrum userà la velocità immediatamente superiore. Per conoscere la velocità correntemente in uso, digitate:

```
PRINT INT (3500000/((PEEK 23747+256*PEEK 23748+2)*26))
```

(Per le velocità più alte, questa formula è inesatta di pochi baud). Se non specificate la velocità di trasmissione e di ricezione, il sistema potrebbe lavorare a velocità imprevedibili. Inizialmente, la velocità viene predisposta per 9600 baud.

## Pilotaggio di una stampante RS232

Prima di connettere una stampante RS232 allo Spectrum, essa dovrà essere predisposta per accettare il protocollo usato dallo Spectrum. Questo si ottiene, in genere, predisponendo opportunamente alcuni switch della stampante. È conveniente predisporre la stampante per la massima velocità di trasferimento (prendetene nota in caso ve ne dimenticaste!) e le opzioni per la struttura dati saranno poste a nessun bit di parità, otto bit di dati ed un bit di stop. Se la stampante dispone della opzione di cambio linea automatico dopo il ritorno carrello, disabilitatela, poiché lo Spectrum produce tutti i necessari controlli.
Predisposta opportunamente la stampante, connettetela per mezzo di un idoneo cavo al connettore a 9 piedini tipo D sul retro dell'interfaccia.
Pilotando una stampante RS232, è consigliabile che siano aperti contemporaneamente i canali "t" e "b": il primo per i testi ed i listati ed il secondo per i codici di controllo. È consigliabile abbinare il canale logico 3 al canale "t" di stampante, in modo da usare gli eventuali comandi LLIST ed LPRINT presenti nei programmi.

```
10 FORMAT "t";1200: REM baud rate
20 OPEN #3;"t": REM testo sul canale logico 3
30 OPEN #15;"b": REM binario sul canale logico 15
40 LLIST: REM list sul canale "t"
50 LPRINT "Misura normale"
60 PRINT #15;CHR$ 14; #3;"Larghezza doppia"
70 CLOSE #3: CLOSE #15
```

Nell'esempio su riportato, il canale logico 15 è usato come file "b" per trasmettere codici di controllo, mentre il canale logico 3 viene sfruttato come canale "t" per i testi ed i listati. Durante la trasmissione RS232, la cornice diventerà nera se il dispositivo che deve ricevere (la stampante) segnala *busy* (occupato) e non è quindi in grado di accettare dati. Lo Spectrum attenderà che il dispositivo si liberi, o fino alla pressione del tasto BREAK. Chiudendo un canale logico RS232 con un comando CLOSE viene trasmesso il CHR$13 che svuota il buffer. Questo carattere non

viene trasmesso sfruttando il comando CLEAR #.
Non è possibile avere più dispositivi RS232 connessi contemporaneamente, anche se è possibile abbinare vari canali logici allo stesso canale fisico. Non è inoltre possibile avere diverse velocità di trasferimento su differenti canali fisici o logici.
L'inconveniente maggiore nel pilotaggio della stampante RS232 è la mancanza del comando TAB. La seguente subroutine in codice macchina inizializzerà il canale logico 3 come canale "t", con una lieve modifica. Esso tiene infatti conto del numero di caratteri inviati, permettendo quindi la piena implementazione della funzione TAB. C'è da chiedersi come mai questa piccola miglioria non sia stata inclusa nella macchina originaria. Essa corregge anche il difetto del software che produce la doppia spaziatura nei listati (ad esempio THEN PRINT). Il codice è stato scritto in modo da stare nell'area di memoria occupata dalla stampante ZX, che non è mai usata quando viene ridefinito il canale logico 3. Notate che né CLOSE #3 né CLEAR # riporteranno il canale logico 3 ad essere connesso alla stampante ZX, né invieranno il codice di ritorno carrello.

**Routine TAB**

```
8997 REM ********************
8998 REM *    RS232 TAB     *
8999 REM ********************
9000 RESTORE 9100: LET c=0
9010 FOR i=23296 TO 23467
9020 READ a: LET c=c+a
9030 POKE i,a
9040 NEXT i
9050 POKE 23540,80: REM larghezza stampa
9055 IF c<>19420 THEN  PRINT "Errore di checksum": STOP
9060 RANDOMIZE USR 23296
9070 RETURN
9100 DATA 42,79,92,1,15,0,9,17
9110 DATA 23,91,115,35,114,1,0,0
9120 DATA 33,245,91,112,35,112,201,254
9130 DATA 32,48,93,254,13,32,26,33
9140 DATA 246,91,203,70,203,134,192,33
9145 DATA 246,91,203,134,43,54,0,62
9150 DATA 13,205,169,91,62,10,195,169
9160 DATA 91,254,23,63,192,17,71,91
9170 DATA 42,81,92,115,35,114,201,50
9180 DATA 15,92,17,79,91,24,241,17
9190 DATA 23,91,205,64,91,58,15,92
9200 DATA 87,33,244,91,150,210,108,4
9210 DATA 35,122,150,213,220,39,91,209
9220 DATA 122,253,150,187,200,71,62,32
```

```
9230 DATA 197,217,215,217,193,16,247,201
9240 DATA 254,165,56,5,214,165,195,16
9250 DATA 12,253,203,188,134,33,59,92
9260 DATA 203,134,254,32,32,2,203,198
9270 DATA 254,128,56,2,62,63,205,169
9280 DATA 91,33,245,91,52,126,43,190
9290 DATA 192,205,39,91,253,203,188,198
9300 DATA 201,207,30,201
```

## Copia dello schermo

Potrebbe interessarvi ottenere una copia su carta di quanto visualizzato su video. Nella forma più semplice, potrete usare il seguente programma che funziona con tutte le stampanti. Il programma legge ogni carattere, inviandolo quindi alla stampante.

```
100 FORMAT "T"; baud: REM valore opportuno
110 OPEN #4;"T"
120 FOR Y=0 TO 21
130 FOR X=0 TO 31
140 LET A$=SCREEN$(Y,X)
150 IF A$="" THEN LET A$=" "
160 PRINT #4;A$;
170 NEXT X
180 PRINT #4
190 NEXT Y
```

La routine lavora scandendo ogni posizione occupabile da un carattere tramite la funzione SCREEN$ e stampando il carattere o, se questo non è riconosciuto, uno spazio (linea 150). Dopo ogni linea di 32 caratteri viene inviato il codice di a capo (linea 180). Anche se un po' rudimentale, questa può essere una routine estremamente utile, grazie anche alla sua totale indipendenza dalla stampante usata. Desiderando invece la copia in alta risoluzione dello schermo, o un listato con caratteri inversi e grafici, come fare? Disponendo di un'opportuna stampante, potete ottenere ciò che desiderate in modo analogo al comando COPY sulla stampante ZX. In BASIC, tuttavia, si tratta di un processo lento, però ne vale la pena. Presento ora due versioni del programma: una per la famiglia di stampanti *Epson* (dotate di interfaccia RS232) ed una per la *Seikosha GP100.*

**Programma EPSON**

```
1000 FORMAT "b";1200: REM cambiare in funzione dell'in
terfaccia
1010 OPEN #3;"b"
1020 LPRINT CHR$ 27;"A";CHR$ 8;
1030 FOR y=175 TO 0 STEP -8
1040 LPRINT CHR$ 27;"K";CHR$ 0;CHR$ 1;
1050 FOR x=0 TO 255
1060 LPRINT CHR$ (128*POINT (x,y)+64*POINT (x,y-1)+32*
POINT (x,y-2)+16*POINT (x,y-3)+8*POINT (x,y-4)+4*POINT
 (x,y-5)+2*POINT (x,y-6)+POINT (x,y-7));
1070 NEXT x
1080 LPRINT CHR$ 13;CHR$ 10;
1090 NEXT y
1100 LPRINT CHR$ 27;"A";CHR$ 12
```

**Programma SEIKOSHA**

```
1000 FORMAT "b";1200: REM cambiare in funzione dell'in
terfaccia
1010 OPEN #3;"b"
1020 LPRINT CHR$ 18
1030 FOR y=174 TO 0 STEP -7
1050 FOR x=0 TO 255
1060 LPRINT CHR$ (POINT (x,y)+2*POINT (x,y-1)+4*POINT
(x,y-2)+8*POINT (x,y-3)+16*POINT (x,y-4)+32*POINT (x,y
-5)+64*POINT (x,y-6)+128);
1070 NEXT x
1080 LPRINT CHR$ 13;CHR$ 10;
1090 NEXT y
1100 LPRINT CHR$ 30
```

Entrambe le routine sfruttano la funzione POINT per calcolare il numero binario che viene trasmesso alla stampante. Notate che viene sfruttato il canale "b" poiché non deve avvenire alcuna conversione di carattere.

## Altri usi della RS232

Il canale "b" può anche essere usato per trasmettere programmi, dati, codice macchina in modo simile ai canali "m", senza però richiedere i nomi dei file. Questa caratteristica è molto utile per trasmettere i programmi sulla linea telefonica, disponendo di un modem. Desiderando tra-

smettere un programma tramite un modem a 300 baud ad un amico all'altro estremo della linea telefonica, questi dovrà innanzitutto porsi in ricezione, digitando

```
FORMAT "b";300: LOAD*"b"
```

e il suo Spectrum attenderà la trasmissione. Mentre voi dovrete battere

```
FORMAT "b";300: SAVE*"b"
```

Il canale "t" dovrà essere usato solo per trasmettere listati o testi. I comandi LOAD*, SAVE*, MERGE* e VERIFY* possono essere usati con la RS232 ed analogamente OPEN # e CLOSE #. I nomi dei file ed i numeri seguenti non sono richiesti con i canali RS232: essi sono accettati dallo Spectrum, ma ignorati. Per esempio, SAVE*"b";5;"nome" viene accettato dallo Spectrum, ma le informazioni extra vengono ignorate.
Un'altra applicazione potrebbe consistere nello sfruttare lo Spectrum come terminale RS232 per trasmettere caratteri ad un altro computer:

```
10 FORMAT "b";baud
20 OPEN #4;"b"
30 PAUSE 0: LET a$=INKEY$
40 IF a$>CHR$ 127 THEN GO TO 30
50 IF a$=CHR$ 6 THEN POKE 23658,8-PEEK 23658: GO TO 30
60 PRINT #4;a$;
70 GO TO 30
```

Il programma esegue una scansione della tastiera (linea 30) ignorando i token (ad esempio STOP) che non sono standard (40) ed usando CAPS LOCK per passare dal modo C (maiuscolo) al modo L (minuscolo) (linea 50). Se si tratta di un carattere valido esso viene inviato al canale logico 4, che lo passa all'altro computer. Alcuni computer possono reagire stranamente ad alcuni dei codici dello Spectrum come INV VIDEO, quindi potrebbe essere necessario qualche altro controllo prima della trasmissione.
È possibile anche usare lo Spectrum come terminale di ricezione, in grado di visualizzare quanto gli arriva tramite l'interfaccia RS232:

```
10 FORMAT "b";baud
20 OPEN #4;"b"
30 LET a$=INKEY$#4
40 IF a$="" THEN GO TO 30
50 IF a$=CHR$ 12 THEN CLS: GO TO 30
60 PRINT a$;
70 GO TO 30
```

Usando la RS232 per l'immissione dati, la pressione del tasto SPACE genererà il BREAK, anche senza la pressione del tasto CAPS SHIFT.
È possibile unire due Spectrum direttamente attraverso la RS232, ma i vantaggi forniti da questa connessione sono pochi, se confrontati alla rete locale, che è più facile da usare con prestazioni simili ma più veloci, lasciando la RS232 libera e disponibile per eventuali periferiche.

## Aggiunte RS232 al programma "Tabella dei canali 1"

Le seguenti linee vanno aggiunte al programma TABELLA DEI CANALI 1 per permettergli il riconoscimento e la gestione dei canali RS232:

**Programma TABELLA DEI CANALI 3**

```
1640 IF F$="T" OR F$="B" THEN  GO TO 3000
2999 REM Canale T/B
3000 PRINT  INVERSE 1;"RS232  ";
3010 IF FN P(D+5)=3130 OR FN P(D+5)=3132 THEN  PRINT "
Testo": GO TO 3040
3020 IF FN P(D+5)=3335 OR FN P(D+5)=3162 THEN  PRINT "
Binario": GO TO 3040
3030 PRINT "(Sconosciuto)"
3040 GO SUB 1800
3050 PRINT "Baud rate:";INT (3500000/((FN P(23747)+2)*
26));" (circa)"
3060 GO TO 1500
```

Entrambi i tipi di canali RS232 hanno in memoria lo stesso identificatore, "T", quindi il riconoscimento del tipo Testo o Binario viene effettuato dalle linee 3010 e 3020. Viene visualizzata inoltre la velocità di trasferimento.

Capitolo 7

# Uso della rete locale

La rete locale è un metodo attraverso cui vari computer possono essere interconnessi in modo da comunicare rapidamente l'uno con l'altro. Sullo Spectrum questa velocità è maggiore di 3K byte per secondo, estremamente superiore ai 19200 baud, massima velocità della RS232. È possibile connettere insieme fino a 64 Spectrum, la qual cosa lascia immaginare un vastissimo numero di applicazioni. Si possono connettere due computer insieme in modo da far loro agevolmente scambiare programmi e dati. Un'altra possibilità d'uso è offerta dallo sviluppo dei programmi, particolarmente in codice macchina. Uno Spectrum può avere in memoria l'Assembler, provvedere alla compilazione del programma in codice macchina e quindi trasmetterlo al secondo Spectrum. Il vantaggio è chiaro: se un errore nel programma blocca il computer, è possibile avere rapidamente una versione modificata, semplicemente ritrasmettendola dal primo con le opportune modifiche.

La rete locale sfrutta canali logici e canali fisici, contraddistinti dall'identificatore di canale "N" (o "n"), iniziale di *network*. Prima di poter sfruttare la rete, dovete assegnare un numero al vostro Spectrum. Per questo esiste il comando

```
FORMAT "n";t
```

dove t è il numero che desiderate attribuirvi, da 1 a 64. All'accensione, lo Spectrum si predispone come numero 1. Se sono presenti in rete solo due Spectrum, non è necessario un FORMAT, in quanto entrambi possono avere il numero 1. Se dimenticate il numero identificatore della vostra stazione, lo potrete facilmente conoscere tramite

PEEK 23749

Per collegare le stazioni, usate i cavetti forniti con ogni Interface 1, connetteteli alle opportune prese, sotto quelle per i registratori a cassette, in modo da formare una catena. Attenzione a non chiudere la catena formando un circolo chiuso: dovete creare un canale al quale tutti gli Spectrum possono accedere e non un loop.

## Trasmissione di programmi

Analogamente a quanto avviene con i Microdrive e l'RS232, programmi e dati possono essere trasmessi sulla rete locale usando comandi simili a quelli sfruttati dall'RS232, ma contraddistinti dall'identificatore "N", seguito dal numero di stazione. Per esempio, se siete la stazione 10 e desiderate trasmettere il vostro programma alla stazione 20, dovrete digitare

SAVE *"n";20

La cornice del vostro schermo diventerà nera nell'attesa che il corrispondente dalla stazione 20 accetti il programma digitando

LOAD *"n";10

Fatto ciò, la cornice di entrambe le stazioni lampeggerà durante il trasferimento del programma. Se la stazione ricevente avesse eseguito il LOAD per prima, quello Spectrum sarebbe rimasto in attesa del vostro comando di SAVE e il programma sarebbe poi stato trasferito perfettamente. Per motivi che vedremo in seguito, è bene che il ricevente digiti LOAD prima che il trasmittente digiti SAVE.
Desiderando la conferma del corretto trasferimento, la stazione ricevente digiterà

VERIFY *"n";10

mentre la trasmittente

SAVE *"n";20

È possibile aggiungere ai comandi tutte le solite funzioni:

SAVE *"n";10 LINE 10
LOAD *"n";20 SCREEN$

## Trasmissione di dati

Per la trasmissione di dati ad un'altra stazione sulla rete, vengono usati canali logici e fisici di tipo "N". Per predisporre un canale logico, usate il comando

OPEN #s;"n";x

con s numero di canale logico (0 – 15) e x numero dell'altra stazione, da 0 a 64 (0 ha una speciale funzione che vedremo in seguito). Quest'operazione crea un buffer, simile a quello usato con i canali "m", ma più piccolo, di 255 caratteri. Quando inizializzate un canale logico di rete, potete scegliere se configurarlo come file di lettura o di scrittura, ricordando che non può essere un file di lettura/scrittura. È la prima operazione che eseguite su di esso che determina il tipo: se eseguite un PRINT # esso diventa un file di scrittura, se leggete da esso (usando INPUT # o INKEY$ #) esso diventa un file di lettura. Analogamente a quanto avviene con i Microdrive, i comandi CLOSE # sono fondamentali per i file di scrittura: se non eseguite il CLOSE, gli ultimi dati presenti nel vostro buffer non saranno trasmessi e, quel che è peggio, la stazione ricevente potrebbe restare in attesa indefinitamente!
Durante le comunicazioni in rete locale (cioè con la cornice lampeggiante) non è necessario premere il tasto CAPS per interrompere: basterà la pressione del tasto SPACE. È bene non accendere o spegnere mai uno Spectrum durante le operazioni di rete.

## Stazione 0 — Broadcasting

Normalmente, durante lo scambio di dati con l'altra stazione, viene eseguito un protocollo di *handshaking*, cioè se la stazione ricevente non è pronta a ricevere, la trasmittente attenderà la sua disponibilità e i dati non verranno quindi persi. Comunque, l'Interface 1 permette il *broadcasting*, usando il numero di stazione 0 che non segue la regola sopra menzionata. Quando trasmettete dati alla stazione 0, essi vengono trasmessi a tutti gli Spectrum presenti in rete, ma il vostro non attenderà il consenso alla trasmissione. Se gli altri computer attendono dati dalla stazione 0, acquisiranno i dati; in caso contrario, questi andranno perduti. Per effettuare trasmissioni con questa procedura, ogni stazione ricevente dovrà digitare

LOAD *"n";0

mettendosi così in stato di attesa del programma. La stazione trasmittente digiterà

```
SAVE *"n";0
```

e tutte le stazioni in ascolto riceveranno il programma. Quando viene sfruttato il protocollo broadcast non è facile sapere chi abbia potuto ricevere quanto trasmesso, ma è possibile conoscere la stazione da cui si è ricevuto, eseguendo

```
PEEK 23759
```

## Programma di gestione della stampante e del Microdrive

Un grande vantaggio della rete locale è che più Spectrum possono condividere le stesse periferiche come stampanti e Microdrive. Presento ora due programmi che mettono in grado ogni stazione *slave* (schiavo) di usare la stampante ed i Microdrive della stazione *master* (capo). Questi programmi sono stati scritti per permettere l'uso più semplice possibile delle periferiche, con comandi come SAVE e LPRINT. Un programma è per la stazione master e l'altro per ogni slave. Per iniziare, ogni stazione dovrà caricare il programma slave, ed il modo più semplice è che la stazione master lo trasmetta in broadcast. Il programma master andrà poi caricato nella stazione con i Microdrive e la stampante ed eseguito. Rilocato il codice macchina la cornice diventa nera ed il sistema è pronto a lavorare. Ogni stazione slave può attribuirsi un qualunque numero di stazione, tranne 32 e 64. Altri programmi possono essere scritti e corretti nelle stazioni slave, purché i numeri di linea siano inferiori a 9000, per evitare di corrompere il programma SLAVE. Se uno slave desidera usare una delle periferiche connesse al master, dovrà digitare GO TO 9000. Viene presentato un menu e l'utente può sfruttare la stampante, caricare e salvare programmi, o farne il catalogo.
Se sceglie l'opzione 1, viene connesso tramite il master a qualunque stampante (ZX o RS232). Può allora usare LPRINT, LLIST, ecc. e, una volta finito, liberare la periferica con CLOSE #3. L'opzione 2 permette di salvare un programma o un file di qualunque tipo sulla cartuccia nella stazione master. L'opzione 3 permette di caricare qualunque file salvato con l'opzione 2 dalla cartuccia del master. L'opzione 4 stampa un catalogo di tutti i programmi di tutti gli utenti e l'opzione 5 termina le operazioni.

**Programma SLAVE**

```
9000 CLS
9010 PRINT '"  1. Richiesta Stampante"
9020 PRINT '"  2. Salvataggio programma"
9030 PRINT '"  3. Caricamento programma"
9040 PRINT '"  4. CATalogo cartuccia"
9050 PRINT '"  5. Fine"
9060 PAUSE 0: LET a$=INKEY$
9070 IF a$="1" THEN  GO TO 9150
9080 IF a$="2" THEN  GO TO 9300
9090 IF a$="3" THEN  GO TO 9400
9100 IF a$="4" THEN  GO TO 9500
9110 IF a$<>"5" THEN  GO TO 9060
9120 GO TO 16000
9147 REM ******************
9148 REM *     Stampa     *
9149 REM ******************
9150 LET a$="print": GO SUB 9200
9160 CLOSE #3: OPEN #3;"n";64
9170 PRINT "Stampante pronta. Usa LPRINT,   LLIST etc.
 Alla fine,digita     CLOSE #3"
9180 GO TO 16000
9197 REM **********************
9198 REM *Broadcast a$ & attesa*
9199 REM **********************
9200 PRINT  INVERSE 1;"Sto chiamando la stazione Maste
r"
9205 CLOSE #4: OPEN #4;"n";0
9210 PRINT #4;a$
9220 CLOSE #4
9230 OPEN #4;"n";64
9240 IF INKEY$#4="" THEN  GO TO 9205
9250 CLOSE #4
9260 PRINT "Collegamento eseguito"
9270 RETURN
9297 REM *****************
9298 REM *      SAVE     *
9299 REM *****************
9300 LET a$="save": GO SUB 9200
9310 INPUT "Nome del file da salvare ?"'; LINE f$
9320 OPEN #4;"n";64
9330 PRINT #4;f$
9340 CLOSE #4
9350 PRINT "Microdrive pronto. Digita      SAVE *""n""
;64 etc"
9360 GO TO 16000
9397 REM *****************
9398 REM *      LOAD     *
9399 REM *****************
```

```
9400 LET a$="load": GO SUB 9200
9410 INPUT "Nome del file da caricare ?"'; LINE f$
9420 OPEN #4;"n";64
9430 PRINT #4;f$
9440 CLOSE #4
9450 OPEN #4;"n";64
9460 INPUT #4;st
9470 CLOSE #4
9480 IF st<>6 THEN  PRINT "File non trovato": STOP
9490 PRINT "Microdrive pronto. Digita        LOAD *""n"
";64 etc"
9495 GO TO 16000
9497 REM *************
9498 REM *    CAT    *
9499 REM *************
9500 LET a$="cat": GO SUB 9200
9510 CLS
9520 PRINT "CATALOGO:"
9530 MOVE "n";64 TO #2
9540 PRINT ' FLASH 1;"Premi un tasto per continuare"
9550 PAUSE 0
9560 GO TO 9000
```

Da notare che il programma MASTER richiede che la routine OPEN #OGNI TIPO e CANALE LOGICO 14−z$ siano presenti dalle linee 7995−8510.
Illustrerò insieme entrambi i programmi, così potrete vedere ciò che accade in entrambe le stazioni.
La prima operazione eseguita dal master consiste nel predisporre i programmi in codice macchina; se desiderate sfruttare come master uno Spectrum da 16K dovete sostituire

```
10 CLEAR 32489
20 LET st=32490: GO SUB 8000
30 LET open=32490: GO SUB 8350
```

Il master deve avere il numero di stazione 64, che gli viene assegnato in linea 100.
La stazione slave deve prima mettersi in contatto con il master, inviando in rete una parola, cui il master risponderà. I file di uno slave vengono immagazzinati sulla cartuccia con 9 lettere di nome e la decima CHR$ N, dove N è il numero della stazione slave. Questo permette di distinguere i programmi delle diverse stazioni slave.
Quando uno slave usa l'opzione 1 per richiedere l'accesso alla stampante, invia ripetutamente in rete PRINT, finché viene ricevuto un carattere dalla stazione 64. Il contatto stabilito dalla subroutine che inizia a linea

9200, che apre il canale logico 4 come broadcast (9205), invia a$ (9210) e chiude il canale logico. Viene quindi attesa una risposta dalla stazione 64 (9230–9240). Se non arriva alcuna risposta, viene ulteriormente trasmesso a$. Ottenuta la risposta, il canale logico viene chiuso (9250). Stabilito il contatto, la linea 9160 apre il canale logico 3 per permettere la trasmissione di dati alla stazione master. Il GO TO 16000 nella 9180 e in altre linee serve a fermare il programma con il messaggio PROGRAM FINISHED.

**Programma MASTER**

```
  1 REM ************************
  2 REM     Stazione Master
  3 REM ************************
 10 CLEAR 65089
 20 LET st=65260: GO SUB 8000
 30 LET open=65090: GO SUB 8350
 40 DEF FN p(p)=PEEK p+256*PEEK (p+1)
100 FORMAT "n";64
110 CLOSE #4
120 OPEN #4;"n";0
130 INPUT #4; LINE a$
140 CLOSE #4
150 LET n=PEEK 23759
160 IF a$="print" OR a$="save" OR a$="load" OR a$="ca
t" THEN  GO TO 180
170 GO TO 120
180 OPEN #4;"n";n
190 PRINT #4
200 CLOSE #4
210 IF a$="save" THEN  GO TO 300
220 IF a$="load" THEN  GO TO 600
230 IF a$="cat" THEN  GO TO 800
237 REM ************************
238 REM * Richiesta Stampante *
239 REM ************************
240 PRINT "Stampante collegata alla        stazione "
;n
250 MOVE "n";n TO #3
260 LPRINT ''''''
270 PRINT "(Stampa completata)"
280 GO TO 110
297 REM **********************
298 REM * Richiesta di SAVE *
299 REM **********************
300 OPEN #4;"n";n
310 PRINT "SAVE in corso alla stazione ";n
```

```
320 DIM f$(10): INPUT #4; LINE f$
330 CLOSE #4
340 GO SUB 500: LET f$(10)=CHR$ n
350 FOR i=1 TO 10
360 POKE USR "a"+i-1,CODE f$(i)
370 NEXT i
380 POKE 23766,d: POKE 23768,5
390 LET a=USR open: CLOSE #5
400 IF a<>1 THEN  ERASE "m";d;f$
410 OPEN #5;"m";d;f$
420 POKE FN p(5*2+23566+8)+FN p(23631)-1+67,4
430 MOVE "n";n TO #5
440 CLOSE #5
450 PRINT "(Salvataggio eseguito)"
460 GO TO 110
497 REM **********************
498 REM *   calcola d da n   *
499 REM **********************
500 LET d=1
510 RETURN
597 REM *********************
598 REM * Richiesta di LOAD *
599 REM *********************
600 OPEN #4;"n";n
610 DIM f$(10): INPUT #4; LINE f$
620 CLOSE #4: LET f$(10)=CHR$ n
630 PRINT "LOAD in corso alla stazione ";n
640 GO SUB 500
670 FOR i=1 TO 10
680 POKE USR "a"+i-1,CODE f$(i)
690 NEXT i
700 POKE 23766,d: POKE 23768,5
710 LET a=USR open
720 OPEN #4;"n";n
730 PRINT #4;a
740 CLOSE #4
750 IF a<>6 THEN  GO TO 770
760 MOVE #5 TO "n";n
770 CLOSE #5
780 PRINT "(Caricamento eseguito)"
790 GO TO 110
797 REM ********************
798 REM * Richiesta di CAT *
799 REM ********************
800 PRINT "CAT in corso alla stazione ";n
805 GO SUB 500
810 LET z$="": CAT #14,d
820 OPEN #4;"n";n
830 LET z$=z$(13 TO )
840 IF LEN z$<10 THEN  GO TO 880
```

```
850 IF z$(10)=CHR$ n THEN  PRINT #4;z$( TO 9)
860 LET z$=z$(12 TO )
870 GO TO 840
880 PRINT #4: CLOSE #4
890 PRINT "(CAT eseguito)"
900 GO TO 110
```

Il programma MASTER legge continuamente stringhe dal canale logico di broadcast, usando le linee 120–140. Controlla poi se si tratta di una richiesta da uno slave (160). In caso negativo, continua l'ascolto dal broadcast (170), in caso contrario calcola il numero dello slave (150) e gli risponde (180–200). Ogni parola di richiesta viene poi controllata eseguendo un appropriato GO TO (210–230). Se la parola PRINT era stata messa in rete, viene visualizzato un opportuno messaggio sul video del master e sfruttato il comando MOVE per trasferire tutti i dati in arrivo dalla stazione slave a qualunque dispositivo collegato al canale logico 3 (linea 250). Quando la stazione slave digiterà CLOSE #3 terminerà il comando MOVE e verrà visualizzato un altro messaggio. La stazione master si metterà nuovamente in ascolto della rete pronta ad esaudire eventuali altre richieste.
Quando uno slave sceglie l'opzione 2 per salvare un suo programma, viene trasmessa in rete la parola SAVE fino alla risposta del master (linea 9300). Viene poi richiesto il nome da attribuire al programma, che viene poi trasmesso al master (9310–9340). Viene visualizzato un messaggio, ed è ora possibile eseguire un SAVE *"n";64 seguito dal tipo di file. Se volete salvare solo programmi, sostituite la linea

```
9350 SAVE *"n";64: GO TO 9000
```

Quando il master legge la parola SAVE da una stazione in rete, apre un canale logico di comunicazione con lo slave e visualizza un messaggio (300–310). La linea 340 sfrutta poi la subroutine in 500 per calcolare il Microdrive corrispondente alla stazione chiamante. In questa versione viene sempre selezionato il drive 1, ma questo può essere poco pratico se c'è un gran numero di stazioni slave. Per esempio, se sono collegate 63 stazioni slave ed il master ha 8 Microdrive collegati, ogni drive potrà essere condiviso da 8 stazioni e la linea 500 sarà

```
500 LET d=1+INT(n/8)
```

Trovato il numero di drive, il decimo carattere del nome del file sarà predisposto per identificare la stazione proprietaria del programma e salvato nell'area destinata ai caratteri grafici definiti dall'utente (350–370). Questo, assieme alla linea 380, predispone tutto quanto è necessario per

sfruttare la routine OPEN #OGNI TIPO sul canale logico 5, in linea 390. Se il file richiesto esiste già, allora viene cancellato (400). La linea 410 apre un file di scrittura dotato del nome richiesto e la 420 inganna il sistema facendogli credere che non si tratta di un file di dati. Il programma trasmesso dallo slave viene scritto nel file, usando MOVE (430). Finita la scrittura, il canale logico viene chiuso e viene visualizzato un messaggio (440–450).
Se la stazione slave sceglie l'opzione 3, di caricamento di un programma, viene trasmessa in rete la parola "LOAD" ed attesa una risposta (9400). Il nome del file viene quindi richiesto e trasmesso al master (9450–9470) e si controlla se il file esiste. Se non esiste, viene introdotto un opportuno messaggio di errore, (9480); se il file esiste, può essere caricato con

```
LOAD *"n";64
```

seguito dagli eventuali attributi richiesti.
Desiderando caricare solo programmi, sostituite la linea

```
9490 LOAD *"n";64: GO TO 9000
```

Quando il master riceve LOAD (220), legge il nome del file dallo slave (600–620) e predispone il decimo carattere in modo da indicare il proprietario. Viene poi calcolato il numero del drive ed il nome del file viene posto nell'area riservata ai caratteri grafici definiti dall'utente. La linea 700 esegue altri POKE, prima di usare OPEN #OGNI TIPO (710) per controllare la presenza del programma richiesto. Il numero ottenuto viene ritrasmesso alla stazione slave. Solo se il file è del tipo corretto, esso viene caricato dal drive e trasmesso allo slave usando MOVE (760). Viene quindi chiuso il canale logico Microdrive e visualizzato un messaggio (770–780).
Se una stazione slave sceglie l'opzione 4 per catalogare i suoi programmi, viene trasmessa la parola CAT, finché non si ottiene la risposta (9500). Viene ripulito lo schermo, visualizzato un titolo e quindi tutti i dati provenienti dalla stazione master (cioè i nomi dei file), usando MOVE (9530). Quando la stazione master riceve una richiesta CAT (230), visualizza un messaggio e calcola il numero del drive (800–805). Con CANALE LOGICO 14–z$, il catalogo viene immagazzinato in z$ (810) e viene aperto un canale logico di comunicazione con lo slave (820). Il titolo viene rimosso dalla stringa (830) e, se è presente un nome della cartuccia, viene testato il decimo carattere. Se esso è CHR$ N, cioè identifica un programma di proprietà dello slave chiamante, allora il nome viene trasmesso allo slave (850). Quando non sono più presenti nomi di file, viene chiuso il canale logico e visualizzato un opportuno messaggio (880–890).
Alcuni utenti potrebbero richiedere ulteriori prestazioni a questi pro-

grammi. Un miglioramento potrebbe consistere nell'uso di una lista di *passwords* (parole d'ordine) per ogni stazione (o in un array o in un file di dati), così che una stazione non possa arbitrariamente attribuirsi il numero di un'altra, distruggendo il lavoro altrui. Se fossero immagazzinati anche i nomi di ogni utente di stazione, nelle stampe potrebbe essere presente una pagina d'intestazione con il nome dell'utente.

## Aggiunte al programma "Tabella dei canali 1" per l'uso con la rete locale

Le seguenti linee vanno aggiunte al programma TABELLA DEI CANALI 1 per permettergli di operare sui canali logici della rete. Il programma visualizzerà il numero di stazione e il numero di destinazione e indicherà l'eventuale uso del broadcast.

**Programma TABELLA DEI CANALI 4**

```
1650 IF F$="N" THEN  GO TO 3500
3499 REM Canale N
3500 PRINT  INVERSE 1;"NETWORK"
3510 GO SUB 1800
3520 PRINT "Stazione numero :";PEEK 23749
3530 PRINT "in transmissione verso";PEEK (D+11);" (in
modo broadcast)" AND PEEK (D+11)=0
3540 GO TO 1500
```

Capitolo

# Il codice macchina e l'Interface 1

# 8

Questo capitolo è stato scritto in particolare per coloro che conoscono il codice macchina dello Z80, ma alcune sue parti possono essere utili per tutti gli utenti.

## La ROM ombra da 8K

In aggiunta alla ROM da 16K del BASIC presente nello Spectrum, l'Interface 1 ha una ROM da 8K che è mappata nella stessa area di memoria, da 0000 a 1FFFH. L'intelligente disegno del meccanismo di banchizzazione rende estremamente agevole al sistema operativo dello Spectrum lo sfruttamento delle routine aggiuntive della ROM ombra, ma rende piuttosto complicata la vita ai programmatori in codice macchina. Prima di approfondire la conoscenza del sistema operativo e della ROM, occorre spiegare il meccanismo di banchizzazione. Per evitare ogni confusione, le locazioni della ROM ombra saranno precedute da un X e, se esistono differenze fra le locazioni delle due ROM, la locazione della vecchia ROM sarà scritta prima della nuova, da cui sarà separata da un trattino.

## Il meccanismo di banchizzazione

Con una eccezione, la ROM ombra viene abilitata solo quando il *program counter* (contatore di programma) dello Z80 assume il valore 0008, cioè

quando viene eseguito RST 8. Nella ROM 16K, questo restart è seguito da un byte di dati che determina quale messaggio di errore debba essere prodotto durante l'esecuzione del programma BASIC (o per produrre un punto interrogativo lampeggiante se la sintassi di una linea è sbagliata). Ogni tentativo di usare uno dei programmi addizionali (ad esempio CAT), o la sintassi estesa (ad esempio LOAD*), genererà, da parte del BASIC, un errore, che sarà trattato cedendo appunto il controllo alla locazione 0008. Se l'Interface 1 è connessa, viene disabilitata la ROM del BASIC ed abilitata quella ombra. Viene poi esaminata la causa dell'errore e se si trattava di un'operazione in BASIC esteso. Se era un errore viene usata la normale routine di gestione dell'errore (nella ROM 16K), se era invece un comando esteso, viene eseguito dalla ROM ombra (eseguendolo o controllandone la sintassi); la ROM ombra viene quindi disattivata dopo che è stato eliminato l'errore che ne ha causato l'abilitazione.
È questo un metodo molto brillante per aggiungere comandi senza cambiare un singolo byte nella ROM 16K dello Spectrum, ma rende piuttosto complicato chiamare le routine della ROM ombra dai programmi utente. Per aiutare il programmatore a superare questo ostacolo, gli autori della ROM ombra hanno usato quelli che loro chiamano *hook code* (codici di interfaccia) che permettono l'accesso ad alcune routine della ROM ombra. Questi codici sono byte di dati che seguono l'istruzione RST 8 e variano da 1BH a 32H incluso (a 34H incluso nella nuova ROM). I codici da FFH a 1AH producono i normali messaggi di errore, mentre i codici superiori a 32 (34 con la nuova ROM) producono HOOK CODE ERROR, non documentato nel manuale. Alcuni di questi codici sono molto utili, altri meno, ma il programmatore deve tener conto dello stato degli interrupt mascherabili, all'ingresso e all'uscita di alcune routine. Se si deve tornare al BASIC (dopo una funzione USR), occorrerà sempre preservare il valore contenuto nella coppia di registri H'L'. Il mancato rispetto di questa precauzione può mandare in tilt il calcolatore in virgola mobile e quindi tutto il sistema.
L'hook code 32H è forse il più utile, anche se il suo uso è ufficialmente riservato alla *Sinclair* per futuri sviluppi. Esso permette ad una routine utente di chiamare la gran parte delle routine della ROM ombra, nonché la abilitazione e la disabilitazione a piacere della stessa ROM ombra. Alcuni lettori potrebbero essere interessati a disassemblare la ROM da 8K, ma, ovviamente, tanto un PEEK quanto un LD A,(HL) saranno in grado di leggere solo la ROM da 16K. Il disassemblato completo è ottenibile tramite il seguente programma, ideato per lo Spectrum 48K:

```
10 CLEAR 32767
20 SAVE *"m";1;"8K ROM"CODE 0,8192
30 LOAD *"m";1;"8K ROM"CODE 32768
```

Quando viene eseguito SAVE*, la ROM ombra è abilitata e salva quindi una copia di se stessa sulla cartuccia. La linea 30 ricarica dalla cartuccia il codice 8000H locazioni sopra quelle di uso normale. Esso pertanto può essere disassemblato partendo dalla locazione 8000, ricordandosi di sottrarre 8000 da tutti gli indirizzi assoluti.
Oltre che da RST 8, la ROM ombra è abilitata anche quando PC=1708H, che è la metà della routine di esecuzione del comando CLOSE #. Questo è causato da un grave errore presente nella routine, che è ben poco adatta a trattare i nuovi tipi di canale usati nell'interfaccia.

## La vecchia e la nuova ROM ombra

Nel marzo 1984, la *Sinclair* ha cambiato la programmazione della ROM ombra fornita nell'Interface 1. Anche se le differenze sono praticamente trasparenti al programmatore BASIC, non lo sono per il programmatore in codice macchina. La nuova ROM è priva di molti degli errori presenti nella vecchia (vedi Appendice C) ed esegue molto più velocemente buona parte delle operazioni con i Microdrive, particolarmente CAT e FORMAT. Purtroppo le routine sono piazzate ad indirizzi diversi e ciò rende il loro uso un po' complesso. Per vedere qual è la ROM presente nella vostra Interface 1, abilitatela (vedrete in seguito come) e controllate il contenuto della locazione X16DA. Nella vecchia ROM è una cella vuota, che contiene FFH, ma nella nuova ROM è l'inizio di un messaggio di copyright e contiene 7FH. Dove possibile, ho cercato di scrivere codici indipendenti dalla ROM.

## Come funzionano i canali logici ed i canali fisici

Esattamente come nel BASIC, i canali logici ed i canali fisici sono disponibili ai programmi in codice macchina. I canali logici sfruttano l'area di memoria STRMS e sono abbinati ai canali fisici che sfruttano l'area CHANS. L'area STRMS è fissa e va da 5C10H a 5C35H incluso e contiene 19 coppie di byte, una coppia per ogni canale logico da FDH a 0FH. Ogni coppia forma uno spiazzamento (o *offset*) a 16 bit, che è pari a 0000 se il canale logico non è associato ad alcun canale fisico. Se lo spiazzamento non è 0 esso punta nell'area CHANS. All'inizializzazione, l'area CHANS consiste di 20 byte, in 4 gruppi di 5, con 5 byte per canale. L'area CHANS inizia alla locazione puntata dalla variabile di sistema CHANS in 5C4FH e finisce 2 byte prima di PROG.
Inizialmente, le informazioni sui canali contengono dati relativi a quattro

canali, con 5 byte riservati ad ognuno, nell'ordine "K", "S", "R" e "P". I primi 16 byte dell'area STRMS contengono i byte mostrati nella seguente tabella con il canale selezionato:

| N° di canale logico | Indirizzo canale logico | Offset del canale fisico | Identificatore di canale |
|---|---|---|---|
| FD | 5C10 | 0001 | "K" |
| FE | 5C12 | 0006 | "S" |
| FF | 5C14 | 000B | "R" |
| 00 | 5C16 | 0001 | "K" |
| 01 | 5C18 | 0001 | "K" |
| 02 | 5C1A | 0006 | "S" |
| 03 | 5C1C | 0010 | "P" |
| 04 | 5C1E | 0000 | nessuno |

Per trovare i dati relativi al canale fisico abbinato ad un canale logico, viene sommato il suo offset a quanto contenuto nella variabile di sistema CHANS (purché lo spiazzamento non sia 0) e quanto ottenuto viene decrementato di 1. Il risultato punta all'inizio dei dati relativi, che sono di almeno 5 byte. La prima coppia di byte punta alla routine ROM di output (uscita dati), la seconda alla routine di input (immissione dati), e il quinto byte è l'identificatore di canale in maiuscolo. Per esempio, il canale "S" è identificato come segue:

```
CHANS+5    DEFW 09F4H
           DEFW 15C4H
           DEFM "S"
```

In effetti eseguendo diverse azioni in funzione dell'identificatore di canale, 09F4H viene usato come routine di output per i canali "K", "S" e "P".

## I canali dell'Interface 1

Per usare una qualunque delle caratteristiche dell'Interface 1, il sistema richiede altri 58 byte per le 9 variabili di sistema, a partire dalla locazione 5CB6H. Quest'area deve essere predisposta prima dell'uso della routine di ROM ombra o di qualunque hook code, nonché prima della creazione di canali (come ad esempio fa la routine CANALE LOGICO 14−z$). Per ottenere questo vengono usate le istruzioni

```
CF RST 08
31 DEFB 31H
```

Se l'Interface 1 non è connessa, viene prodotto il messaggio di errore "i" (NON EXISTENT). Se l'interfaccia è presente vengono inseriti i 58 byte. Le variabili sono illustrate nell'Appendice A.
Le variabili aggiuntive di sistema vengono normalmente create quando:

1. Avviene un errore (di sintassi o durante l'esecuzione)
2. Viene introdotto o eseguito ogni comando esteso del BASIC
3. Viene eseguito un comando CLOSE #

Inizialmente la variabile VECTOR in 5CB7H, contiene 01F0H, ma questo valore può essere cambiato per permettere l'aggiunta di nuovi comandi al BASIC. Solo questo aspetto dell'Interface 1 fornirebbe materia sufficiente per un libro; noi ci limiteremo a fornire alcuni dettagli nel prossimo capitolo.
La variabile IOBORD in 5CC6H viene usata per determinare il colore della cornice durante certe operazioni di I/O, cioè RS232, rete locale e durante la scrittura su una cartuccia. Se preferite un certo colore della cornice, vi basterà fare POKE 23750, colore richiesto. Se, inoltre, non gradite la cornice lampeggiante, potrete fare

```
POKE 23750,INT(PEEK 23624/8)
```

che stabilisce come colore di I/O, il colore normale. I canali aggiuntivi di interfaccia "M", "N", "T" e "B" usano una struttura estesa per immagazzinare i loro dettagli nell'area CHANS.

## Canale "M"

Occupa 595 byte nell'area CHANS ed è indirizzato dal registro IX nella ROM ombra. La locazione dei byte è la seguente (tutti gli offset sono in decimale):

| | | |
|---|---|---|
| 0 | DEFW 8 | routine output |
| 2 | DEFW 8 | routine input |
| 4 | DEFM "M" | identificatore di canale |
| 5 | DEFW 11D8H/12B3H | output sulla ROM ombra |
| 7 | DEFW 1122H/11FDH | input sulla ROM ombra |
| 9 | DEFW 595 | lunghezza di questa area CHANS |
| 11 | CHBYTE | posizione corrente del buffer (0–512) |
| 13 | CHREC | posizione del record nel file (0–255) |
| 14 | CHNAME | 10 byte di nome del file |

| | | |
|---|---|---|
| 24 | CHFLAG | bit 0=0 — read file<br>0=1 — write file<br>bit 1−7 non usati (tutti posti ad 1) |
| 25 | CHDRIVE | numero del drive (1−8) |
| 26 | CHMAP | locazione della mappa del drive |
| 28 | HPREAM | 12 byte di preambolo dell'intestazione |
| 40 | HDFLAG | bit 0=1 per segnalare l'intestazione<br>bit 1−7 non usati |
| 41 | HDNUM | numero di settore (0−255) |
| 42 | | non usato |
| 44 | HDNAME | 10 byte con il nome della cartuccia |
| 54 | HDCHK | checksum dell'intestazione |
| 55 | DPREAM | 12 byte di preambolo dati |
| 67 | RECFLG | bit 0=0 se non intestazione<br>bit 1=0 non EOF<br>bit 1=1 EOF<br>bit 2=0 PRINT file<br>bit 2=1 non PRINT file<br>bit 3−7 non usati (tutti posti a 0) |
| 68 | RECNUM | numero del record (0−255) |
| 69 | RECLEN | numero di byte nel record (0−512) |
| 71 | RECNAM | 10 byte di nome del file |
| 81 | DESCHK | checksum delle variabili da RECFLG a DESCHK−1 |
| 82 | CHDATA | primo dei 512 byte di buffer |
| 593 | | ultimo byte del buffer |
| 594 | DCHK | checksum del buffer |

Ogni Microdrive in uso ha inoltre riservati 32 byte nell'area delle mappe dei Microdrive, da 5CF0H a CHANS−1. Ogni blocco di 32 byte è la *bitmap* di ogni settore sulla cartuccia. Se un bit è posto a 1, allora il settore non è disponibile, perché contiene dati, o è danneggiato, o non esiste.

## Canale "N"

Occupa 276 byte nell'area CHANS ed è indirizzato dal registro IX nella ROM ombra. La locazione dei byte è la seguente:

| | | |
|---|---|---|
| 0 | DEFW 8 | |
| 2 | DEFW 8 | |
| 4 | DEFM "N" | identificatore di canale |

| | | |
|---|---|---|
| 5 | DEFW 0D6CH/0EA9H | routine ombra di output |
| 7 | DEFW 0D0CH/0DA9H | routine ombra di input |
| 9 | DEFW 276 | lunghezza di quest'area CHANS |
| 11 | NCIRIS | numero della stazione destinataria (0–64) |
| 12 | NCSELF | numero di questa stazione (0–64) |
| 13 | NCNUMB | numero di blocco (0–65535) |
| 15 | NCTYPE | tipo di pacchetto: 0–dati, 1–fine del file |
| 16 | NCOBL | byte del blocco dati (0 se output) |
| 17 | NCDCS | checksum dei dati |
| 18 | NCHCS | checksum dell'intestazione |
| 19 | NCCUR | posizione del buffer di input |
| 20 | NCIBL | numero di byte nel buffer di input (0 se output) |
| 21 | NCB | inizio dei 255 byte del buffer di input |

## Canali "T" e "B"

Ognuno di questi occupa 11 byte nell'area CHANS, il minimo per un canale diverso da quelli standard. La locazione è:

| | | |
|---|---|---|
| 0 | DEFB 8 | |
| 2 | DEFB 8 | |
| 4 | DEFB "T" (vecchia ROM)/"T" o "B" (nuova ROM) | identificatore |
| 5 | DEFW 0C3CH/0C3AH (T) o 0C5AH/0D07H (B) | output ombra |
| 7 | DEFW 0B6FH/0B76H (T) o 0B75H/0B7CH (B) | input ombra |
| 9 | DEFW 11 | lunghezza di quest'area CHANS |

Vengono predisposti dei canali temporanei quando si usa SAVE * ecc., MOVE, FORMAT, ERASE, dal BASIC o dal codice macchina. Questi sono simili ai canali ora descritti, con tre importanti differenze:

1. L'identificatore ha il bit più significativo posto a 1
2. Non sono mai connessi ad un canale logico
3. Vengono cancellati al completamento del comando, o se avviene un errore
4. Sono sempre alla fine di CHANS; un canale permanente non dovrebbe mai risiedere in memoria a locazioni più alte di qualunque canale temporaneo

Con canali diversi da quelli standard, il numero di byte minimo nell'area

CHANS è 11 per canale, quanti sono usati da "T" e "B". Le prime due coppie di byte, usualmente i puntatori alle routine di output e input, sono entrambe 0008 e producono l'abilitazione della ROM ombra. La ROM ombra rileva che è stata richiesta una routine di canale e usa la coppia di byte locata 5 byte oltre nella area CHANS selezionata, cioè la routine della ROM ombra. Alla fine della routine, la ROM ombra si disabilita. La coppia di byte in nona e decima posizione è un numero a 16 bit che informa il sistema sul numero di byte occupati dal canale nell'area CHANS. Questa coppia di byte è estremamente importante, in quanto viene usata in caso di errore per scandire l'area CHANS alla ricerca di canali temporanei da chiudere. Se essi mancano, o sono errati, il sistema si bloccherà al primo errore.

## Uso dei canali logici

Per usare un canale logico per input o output, va usata la routine STRM_OPEN in 1601H. (Non ha niente in comune con il comando BASIC OPEN #). In ingresso alla routine, l'accumulatore deve contenere un valido numero di canale logico (da FDH a 0FH). Da esso, viene trovato il relativo spiazzamento nell'area STRMS e viene prodotto l'errore INVALID STREAM se esso è 0; se non è 0, viene calcolata l'opportuna locazione in CHANS e il suo valore viene immagazzinato nella variabile di sistema CURCHL in 5C51H.
Per emettere caratteri sul canale logico corrente, ponete il codice nell'accumulatore e chiamate la routine OUTPUT in 0010H tramite l'istruzione RST 10. I registri BC, DE e HL non vengono alterati.
Per ottenere l'input di caratteri dal canale logico corrente, SET 4, (IY+124), è chiamata la routine INPUT in 15E6H. I registri BC, DE e HL non vengono alterati. Al ritorno, il flag di carry sarà uguale ad 1 se l'accumulatore contiene un carattere valido e il flag di zero sarà ad 1 se non è stato letto alcun carattere. In caso contrario carry e zero saranno entrambi posti a 0, se si è verificata una situazione di END OF FILE.

## Uso dei canali fisici

Non esiste, purtroppo, in codice macchina alcun semplice equivalente al comando OPEN #. Durante la sperimentazione, il canale logico richiesto può essere aperto dal BASIC e usato dal codice macchina, ma questo è probabilmente poco soddisfacente per un buon uso. Gli equivalenti in codice macchina per ogni tipo di comando OPEN # vengono forniti nel se-

guito del capitolo, nelle sezioni relative. Il codice macchina, equivalente a CLOSE # è:

```
LD A, stream
RES 1, (IY+124)       ;segnala non CLEAR #
LD HL,1718H           ;routine CLOSE nella ROM ombra
LD (HD_11),HL
RST 8
DEFB 32H
RET
```

Questo sfrutta l'hook code 32H per chiamare la routine CLOSE # della ROM ombra.

## Le routine della ROM ombra

Segue ora un elenco delle routine ROM ombra necessarie per sfruttare le capacità dell'Interface 1. Vengono forniti dettagli su come usare le routine (generalmente tramite gli hook code) e sulle condizioni dei registri e degli interrupt in ingresso e in uscita. Vengono anche forniti i registri i cui valori sono alterati dalle routine, nonché le locazioni di memoria in cui risiedono le routine stesse.

### LE ROUTINE DEL MICRODRIVE

OPEN_M: crea un canale temporaneo "M" nell'area CHANS

| | |
|---|---|
| Per usarla: | RST 8<br>DEFB 22H |
| Entrata: | interrupt abilitati |
| Uscita: | IX=inizio dell'area, HL=offset del canale logico (=IX−CHANS+1), INTERRUPT disabilitati, motore acceso |
| Registri modificati: | AF, BC, DE, HL, B'C', D'E', H'L', IX |
| Locazioni: | X1B29H/1B05H |

Prima dell'uso, la variabile di sistema D_STR1 deve contenere il numero di drive, N_STR1 la lunghezza del nome del file e T_STR1 deve puntare all'inizio del nome del file. Viene prima inserita un'area di 595 byte alla fine dell'area CHANS (spazio di memoria permettendo) in cui vengono co-

piati i relativi attributi. Se il drive scelto non possiede già un'area Map, questa viene creata e riempita di FF. Il drive viene quindi posto in funzione e viene cercato il nome del file sulla cartuccia. Se esso viene trovato, viene letto il primo settore e il bit 0 di CHFLAG viene posto a 0; in caso contrario esso viene posto ad 1 per indicare un file di scrittura. Se è un file di lettura, il bit 0 di IX+25 (CHFLAG) viene posto a 0 se è un file tipo PRINT; in caso contrario viene posto ad 1 (per programmi, byte, ecc.). Non viene controllata la linguetta di protezione contro la sovrascrittura. Per fare questo, dopo la routine, eseguite

```
IN A,(EFH)
AND 1                   ;zero se protetto
```

Notate che il motore del drive non viene spento da questa routine e che il canale creato è temporaneo. Per renderlo permanente e abbinarlo ad un canale logico eseguite:

```
LD A,stream
ADD A,A
LD HL,5C16H
LD E,A
LD D,0
ADD HL,DE
PUSH HL                 ;salva la locazione del canale logico
RST 8
DEFB OPEN_M
PUSH HL                 ;salva lo spiazzamento del canale logico
XOR A
RST 8
DEFB MOTOR              ;spegne il motore
POP DE
POP HL
LD (HL),E
INC HL
LD (HL),D               ;salva i nuovi dati in STRMS
RES 7,(IX+4)            ;lo rende permanente
RET
```

MAKE_M: crea un'area "M" di 595 byte in CHANS

```
Per l'uso:      vecchia ROM           nuova ROM
                LD HL,0FE8H           RST 8
                LD (HD_11),HL         DEFB 2BH
                RST 8
                DEFB 32H
```

| | |
|---|---|
| Entrata: | interrupt abilitati |
| Uscita: | IX=inizio dell'area, HL=IX−CHANS+1, interrupt abilitati |
| Registri modificati: | AF, BC, DE, HL, B'C', D'E', H'L', IX |
| Locazione: | X0FE8H/10A5H |

Prima dell'uso, D_STR1 deve contenere il numero del drive, N_STR1 la lunghezza del nome del file, T_STR1 l'inizio. Un'area di 595 byte viene inserita alla fine di CHANS (memoria permettendo) e vi vengono copiati gli attributi. Se il drive non possiede già una Map, essa viene creata e riempita di FFH. Nella vecchia ROM, non può essere usato l'hook code 2BH, poiché esso è erroneamente lo stesso di 22H.

MOTOR: accende il motore di un drive o spegne i motori di tutti i drive

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 21H |
| Entrata: | A=1−8 per accendere il motore di un drive, A=0 per spegnerli tutti |
| Uscita: | interrupt disabilitati se A< >0, interrupt abilitati se A=0 |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X17F7H/1532H |

Se l'accumulatore è diverso da 0, il motore del relativo Microdrive viene acceso, tutti gli altri spenti e gli interrupt vengono disabilitati. Se A è uguale a zero, vengono spenti i motori di tutti i drive e abilitati gli interrupt. Se il drive selezionato non è collegato, o se manca la cartuccia, o se questa non è formattata, viene generato l'errore MICRODRIVE NOT PRESENT.

CLOSE_M: chiude un canale "M"

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 23H |
| Entrata: | IX=inizio dell'area del canale "M" |
| Uscita: | interrupt abilitati |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X12A9/138EH |

Se il canale scelto era un file di scrittura, viene scritto il contenuto del

buffer, viene spento il motore del drive e viene liberata l'area di 595 bit. Viene anche liberata l'area Map, salvo che non sia usata da un altro canale.

ERASE_M: cancella un file su Microdrive

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 24H |
| Entrata: | nessuna |
| Uscita: | interrupt abilitati |
| Registri modificati: | AF, BC, DE, HL, B'C', D'E', H'L', IX |
| Locazione: | X1D6EH/1D79H |

Prima dell'uso, le variabili di sistema D_STR1, T_STR1 e N_STR1 vanno predisposte con il numero del drive selezionato e con la stringa del file da cancellare. Viene creato un canale "M" temporaneo e viene ricercato il file sulla cartuccia. Se trovato, esso viene cancellato (nel caso di copie multiple solo una viene cancellata), purché sia presente la linguetta di protezione contro la sovrascrittura; se non è presente, viene generato un errore. Alla fine il motore viene spento e il canale temporaneo cancellato.

RECLAIM_M: libera un'area "M" di 595 byte

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 2CH |
| Entrata: | IX=inizio dell'area del canale "M" |
| Uscita: | nessuna |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X10C4H/119FH |

Vengono liberati 595 byte dell'area e viene chiuso ogni canale logico che punta ad essa. L'area Map di 32 byte viene liberata solo se non è usata da un altro canale.

CAT: esegue il catalogo di una cartuccia di Microdrive

Per l'uso:

```
LD HL,CATANY
LD (HD_11),HL
RST 8
DEFB 32H
```

```
CATANY          LD HL,(04B0H)
                INC HL
                LD E,(HL)
                INC HL
                LD D,(HL)
                EX DE,HL
                JP (HL)
```

Entrata: interrupt abilitati
Uscita: nessuna
Registri modificati: AF, BC, DE, HL, A'F', B'C', D'E', H'L', IX
Locazione: X1C58H/1C52H

Prima dell'uso, S_STR1 deve contenere il numero di canale logico per il catalogo e D_STR1 il numero del drive. Poiché non esiste hook code per CAT, viene sfruttato il codice 32H insieme con HD_11 per chiamare la routine CATANY con la ROM ombra abilitata. CATANY calcola dove sia la routine CAT esaminando la routine di sintassi che è allo stesso posto in entrambe le ROM. Innanzitutto, viene aperto il canale logico scelto (usando 1601H), viene quindi creato un canale "M" temporaneo. Vengono esaminati tutti gli *header* (intestazioni sulla cartuccia) e il nome del file immagazzinato nel buffer di 512 byte nell'area CHANS, salvo che esso sia già là, o che inizi con CHR$ 0. Esso viene inserito in ordine alfabetico nella corretta posizione del buffer. Eseguita la lettura di tutta la cartuccia, o letti 50 nomi di file, il drive viene spento e il nome della cartuccia visualizzato sul canale logico prescelto. Ogni nome di file viene poi estratto dal buffer e visualizzato; viene infine visualizzato il numero di Kbyte libero. Questo ultimo numero è dato dalla metà del numero di bit posti a zero nella relativa area Map. Viene infine liberata l'area "M".

## LE ROUTINE DI SERVIZIO DELLA CARTUCCIA

Prima di spiegare le altre routine del Microdrive, occorre illustrare la struttura dei dati sul nastro. Una cartuccia è divisa in circa 180 settori, ognuno di 512 byte di dati. La routine è in grado di gestire 256 settori, ma il settore zero e quelli sopra 180 (circa) o non esistono, o non sono mai usati. Inoltre, due o tre settori non sono utilizzabili, essendo in corrispondenza della giunzione dell'anello del nastro.
Prima di ogni settore, è presente un header, che consiste di 12 byte di preambolo (10 zeri e due FF), quindi 15 byte delle variabili

HDFLAG – HDCHK. Ogni header ha un numero diverso (HDNUMB) da 1 a circa 180, immagazzinato sequenzialmente sulla cartuccia.
Dopo l'header abbiamo il settore vero e proprio che consiste di 12 byte di preambolo (come sopra), 15 byte di variabili (RECFLG – DESCHK), quindi 512 byte di dati e infine un byte di checksum (DCHK). Un settore non è usato se il bit 1 di RECFLG e il bit 1 di RECLENhi (IX+70) sono entrambi posti a zero.
Poiché molti file sono troppo grandi per stare dentro un unico settore, vengono divisi in record di 512 byte, ognuno dei quali viene numerato sequenzialmente partendo da 0. Se un settore contiene meno di 512 byte, il bit 1 di RECFLG viene posto ad 1 per mostrare che si tratta dell'ultimo settore. I file di tipo non PRINT, come i programmi, immagazzinano i loro attributi nei primi 9 byte del record numero zero. Questi attributi sono HD_00 – HD_11, spiegati nell'Appendice A. I rimanenti hook code dei Microdrive sono i seguenti:

READ_P: legge un record da un file di tipo PRINT

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 27H |
| Entrata: | IX=inizio dell'area del canale "M" |
| Uscita: | interrupt disabilitati, motore acceso |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X1A17H/1F0BH |

Prima dell'uso di questa routine, CHDRIV deve contenere il numero del drive, CHREC il numero del record richiesto e CHNAME il nome del file. Innanzitutto viene acceso il drive selezionato, SECTOR posto a 04FB (5 giri completi del nastro) e viene effettuata la ricerca del record scelto. Se viene trovato ed è un file di tipo PRINT, il settore viene letto nel buffer e viene restituito il controllo alla routine chiamante. Se esso non è di tipo PRINT, l'area viene liberata e viene generato l'errore WRONG FILE TYPE. Se il record non viene trovato dopo 5 completi giri del nastro, viene generato l'errore FILE NOT FOUND e l'area liberata, se era temporanea.

READ_NP: legge il record successivo in un file di tipo PRINT

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 25H |
| Entrata: | IX=inizio dell'area "M" |
| Uscita: | interrupt disabilitati, motore acceso |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X1A09H/1EFDH |

Questa routine è simile a READ_P; CHDRIV deve contenere il numero del drive e CHNAME il nome del file. Viene innanzitutto controllato il bit 1 di RECFLG per vedere se il record correntemente in memoria sia l'ultimo, in tal caso, viene segnalato END OF FILE. In caso contrario, viene incrementato il contenuto di CHREC e il controllo viene ceduto a READ_P.

READ_S: legge il prossimo settore

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 29H |
| Entrata: | IX=inizio dell'area "M", interrupt disabilitati, motore acceso |
| Uscita: | interrupt disabilitati, motore acceso |
| Registri modificati: | AF, BC,DE, HL |
| Locazione: | X1A86H/1F7AH |

I prossimi header e settore sulla cartuccia vengono trasferiti nell'area di canale. Se si tratta di un file di tipo PRINT si ritorna con il flag di carry posto a zero. In caso contrario, o in caso di errore di checksum, viene cancellato il contenuto del buffer e si torna con il carry posto ad 1.

READ_R: legge random un settore PRINT

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 28H |
| Entrata: | IX=inizio dell'area "M", interrupt disabilitati, motore acceso |
| Uscita: | interrupt disabilitati, motore acceso |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | H1A4BH/1F3FH |

Viene cercato sulla cartuccia un settore numero CHREC, che, se trovato, viene copiato nel buffer. Se si tratta di un file di tipo PRINT, il flag di carry viene posto a 0 e il controllo ritorna alla routine chiamante. In caso contrario, viene cancellato il contenuto del buffer e posto ad 1 il carry. Se il settore scelto non è trovato dopo un giro completo del nastro, viene generato l'errore FILE NOT FOUND.

SCAN_M: legge un header dalla cartuccia (solo nella nuova ROM)

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 33H |
| Entrata: | IX=inizio dell'area "M", interrupt disabilitati, motore acceso |
| Uscita: | interrupt disabilitati, motore acceso |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X1FE4 (solo nuova ROM) |

Si legge il nastro via via che questo passa davanti alla testina del Microdrive fino a che viene trovato un header valido, che viene copiato nelle celle HDFLAG – DESCHK incluse. In caso di errore di checksum, o se il nome del file del settore inizia con CHR$ 0, l'intera area viene cancellata e il flag di carry posto ad 1, in caso contrario viene posto a 0. Per leggere l'header di settori adiacenti, il codice presente a due chiamate successive a questa routine deve essere eseguito in meno di 30 secondi.

WRITE_S: scrive serialmente un settore

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 26H |
| Entrata: | IX=inizio dell'area "M" |
| Uscita: | interrupt abilitati, motore spento |
| Registri modificati: | AF, BC,DE, HL |
| Locazione: | X11FFH/12DAH |

Prima della chiamata predisponete opportunamente le variabili CHBYTE, CHREC, CHNAME, CHDRIVE e il buffer. Viene innanzitutto acceso il motore del drive scelto e controllato se la cartuccia è piena. Il nome del file viene quindi copiato da CHNAME a RECNAM, e CHBYTE e CHREC in RECLEN e RECNUM. Vengono calcolati i checksum DESCHK e DCHK e l'area da RECFLG a DCHK viene scritta sul primo settore libero sulla cartuccia (purché essa non sia protetta dalla sovrascrittura). Il relativo bit corrispondente della Map viene posto a 1, CHBYTE viene azzerato e CHREC incrementato. Il motore viene infine spento.

WRITE_R: scrive random un settore

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 2AH |
| Entrata: | IX=inizio dell'area "M", interrupt disabilitati, motore acceso |
| Uscita: | interrupt disabilitati, motore acceso |

| | |
|---|---|
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X1A91H/1F85H |

Prima della chiamata, CHREC deve contenere il valore richiesto e CHNAME il nome del settore da scrivere. Il settore numero CHREC viene poi ricercato sulla cartuccia. Se non viene trovato, viene generato l'errore FILE NOT FOUND. In caso contrario, viene controllata la linguetta di protezione contro la sovrascrittura. Se essa è presente, il settore e i dati (RECFLG-DCHK) vengono scritti sulla cartuccia e il relativo bit nella Map viene posto a 1. Notate che la Map non viene controllata prima della scrittura, quindi il controllo dovrebbe essere fatto dall'utente se lo desidera.

**LE ROUTINE RS232**

Prima dell'uso di qualunque routine RS232, devono essere presenti le variabili aggiuntive di sistema e va selezionato un opportuno baud rade (velocità di trasferimento). Alla prima operazione provvede l'hook code 31H, per la seconda occorre alterare la variabile BAUD in 5CC3H.

232IN: legge un byte dalla RS232

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 1DH |
| Entrata: | nessuna |
| Uscita: | carry=1 se è stato letto un byte, A=byte, interrupt abilitati |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X0B81H/0B88H |

Legge un byte dalla porta RS232, ma l'operazione fallisce in caso di attesa troppo lunga, o pressione del tasto SPACE (vecchia ROM) o BREAK (nuova ROM). Il flag di carry è posto a 1 se è stato letto un byte.

232OUT: emette un byte dall'RS232

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 1EH |
| Entrata: | interrupt abilitati |
| Uscita: | A=codice del byte |

| | |
|---|---|
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X0C5AH/0D07H |

Un byte viene inviato attraverso la porta alla corretta velocità e con il necessario protocollo. In caso di BREAK durante la trasmissione viene generato un errore.
Se volete ottenere l'equivalente del canale "T" per la RS232, usate l'hook code 32H per chiamare X0C3CH (vecchia ROM) o X0C3AH (nuova ROM) col codice del carattere posto in A. Notate che la nuova ROM usa due variabili di sistema in più, durante la gestione dei canali "T": 5CB0H per la posizione dei cursori e 5CB1H per l'ampiezza della stampante. Questi sono inizializzati rispettivamente fra 0 e 80.

OPEN_R: apre un canale RS232

| | | |
|---|---|---|
| Per l'uso: | vecchia ROM | nuova ROM |
| | LD HL,0B13H | LD HL, 0B17H |
| | LD (HD_11),HL | LD (HD_11),HL |
| | RST 8 | RST 8 |
| | DEFB 32H | DEFB 32H |
| Entrata: | interrupt abilitati | |
| Uscita: | DE=inizio della nuova area CHANS | |
| Registri modificati: | AF, BC, DE, HL | |
| Locazione: | X0B13H/0B17H | |

Questa routine crea un'area di 11 byte alla fine di CHANS per un canale "B" o "T". Normalmente essa crea un canale "T", salvo che L_STR1 contenga "B", nel qual caso creerà un canale "B". La nuova ROM possiede un hook code aggiuntivo OPEN_B, 34H, che apre un canale "B".

## LE ROUTINE DI RETE LOCALE

La rete locale è implementata sullo Spectrum da una connessione continua fra ogni singola stazione. Può avvenire solo una comunicazione alla volta e i dati vengono trasmessi serialmente. Essi sono divisi in pacchetti numerati, ognuno lungo al massimo 255 byte. Un pacchetto viene trasferito nel modo seguente: innanzitutto, la stazione trasmittente controlla se la rete sia occupata (*busy*), in tal caso, attende che si liberi. A rete finalmente libera, la stazione trasmittente emette 8 byte di intestazione, composti dalle variabili di sistema NTDEST–NTCHC. Questo iter contiene varie informazioni sui dati che lo seguiranno e, in particolare, il numero

della stazione trasmittente, il numero della stazione destinataria e due checksum. Salvo che lavori in modo broadcast, la stazione trasmittente attende un assenso dalla ricevente, sotto forma di un singolo byte con il numero della stazione ricevente. Se il byte non viene ricevuto, l'header viene trasmesso di nuovo. Se opera in broadcast, non viene atteso alcun byte di risposta e i byte vengono trasmessi anche senza assenso. La sezione dati non viene trasmessa se vuota, cioè se NTLEN=0, in caso contrario viene emesso l'opportuno numero di byte. Si attende poi un altro byte di risposta (salvo che non si sia in modo broadcast) e, in caso di mancata ricezione, il processo si ripete, ritrasmettendo l'header. Esistono 4 hook code per l'uso del Network, dei quali uno non è purtroppo usabile con la vecchia ROM.

OPEN_N: apre un canale temporaneo "N"

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 2DH |
| Entrata: | nessuna |
| Uscita: | IX=DE=(CURCHL)=inizio dell'area "N" in CHANS |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X0EA9H/0F46H |

Prima della chiamata, D_STR1 deve contenere il numero della stazione destinataria e NSTAT il numero scelto per la propria stazione. Viene poi creata un'area di 256 byte alla fine dell'area CHANS, in cui vengono immagazzinati i dati. Il numero della stazione destinataria viene copiato da D_STR1 a NCIRIS, il numero di stazione da NSTAT a NCSELF e l'area, da NCNUMB all'ultimo carattere nel buffer, viene riempita di zeri. Il canale viene reso temporaneo ponendo a 1 il bit 7 dell'identificatore di canale. La variabile CURCHL viene fatta puntare all'inizio dell'area così creata. Per ottenere un canale permanente di rete locale, usate il seguente codice (dopo aver predisposto D_STR1 e NTSTAT):

```
LD A, stream
ADD A,A
LD HL, 5C16H
LD E,A
LD D,0
ADD HL,DE          ;HL=locazione idonea in STRMS
PUSH HL            ;la salva
RST 8
DEFB OPEN_N        ;apre un temporaneo "N"
```

```
RES 7, (IX+4)       ;lo rende permanente
LD HL, (CHANS)
EX DE, HL
AND A
SBC HL,DE
INC HL              ;HL=offset del canale logico
POP DE              ;DE=locazione STRMS
EX DE, HL
LD (HL), E          ;salva l'offset in STRMS
INC HL
LD (HL), D
RET
```

CLOSE_N: chiude un canale "N"

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 2EH |
| Entrata: | (CURCHL)=inizio dell'area "N" in CHANS |
| Uscita: | interrupt abilitati |
| Registri modificati: | AF, BC, DE, HL, IX |
| Locazione: | X1A24H/1F18H |

Se il canale fisico è un file di scrittura (cioè se NCOBL< >0) quanto è ancora nel buffer viene trasmesso in un pacchetto marcato EOF. I 256 byte dell'area vengono poi liberati, ma non viene chiuso alcun canale logico ivi abbinato. L'area "N" deve essere temporanea, o essere l'ultimo canale dell'area CHANS, in caso contrario, potrebbe alterare altri canali logici o fisici.

WRITE_N: trasmette un pacchetto sulla rete locale

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 30H |
| Entrata: | A=0 per dati, 1 per EOF; IX=inizio dell'area "N" |
| Uscita: | interrupt abilitati, A=numero di stazione destinataria (Z se in modo broadcast) |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X0DB2H/0E4FH |

Prima della chiamata, vanno opportunamente predisposte le variabili NCOBL, NCNUMB, il contenuto del buffer e l'area header (esclusi i

checksum). In entrata, l'accumulatore dovrebbe contenere 0, o 1 se il pacchetto è l'ultimo della sequenza, cioè un segnalatore di EOF. Il valore viene immagazzinato in NCTYPE, il colore della cornice diventa IOBORD. Il checksum del contenuto del buffer viene immagazzinato in NCDCS e il checksum di NCIRIS–NCDCS viene immagazzinato in NCHCS. Gli interrupt vengono disabilitati e viene inviato il pacchetto di dati. Quando il pacchetto è stato ricevuto (se non in modo broadcast), viene incrementato NCNUMB, ripristinato il colore originale di cornice ed abilitati gli interrupt.

READ_N: legge un pacchetto dalla rete locale (solo nuova ROM)

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 2FH |
| Entrata: | IX=inizio dell'area "N" |
| Uscita: | interrupt abilitati |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | (X1A31H)/1F25H |

Questo hook code legge un pacchetto di byte dalla rete locale. Il flag di carry è posto a 0 se l'operazione è stata portata a termine con successo, in caso contrario esso è posto a 1, ma viene alterato nella routine della vecchia ROM. Un modo alternativo per leggere caratteri dalla rete locale è per mezzo della routine INPUT presente nella ROM 16K, all'indirizzo 15E6. Ricordate di salvare CURCHL se usate altri canali fra una chiamata e l'altra a questa routine.

## HOOK CODE VARI

PAUSE: attende che sia premuto un tasto

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 1BH |
| Entrata: | nessuna |
| Uscita: | interrupt abilitati, A=codice del tasto |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X19D9H/1ECDH |

Gli interrupt vengono abilitati e ogni 50-esimo di secondo viene chiamata la routine di scansione tastiera della ROM 16K, fino alla pressione di un tasto. Il valore del tasto viene letto dalla variabile LAST_K.

PRINT: visualizza un carattere sullo schermo

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 1CH |
| Entrata: | A=codice del carattere, interrupt abilitati |
| Uscita: | nessuna |
| Registri modificati: | AF, BC, DE, HL, A'F', B'C', D'E' |
| Locazione: | X19ECH/1EE0H |

La variabile SCR_CT viene posta a FF per abilitare lo scroll automatico, viene aperto il canale logico FEH (canale "S") e il carattere visualizzato.

LPRINT: stampa un carattere sulla stampante ZX

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 1FH |
| Entrata: | A=codice del carattere, interrupt abilitati |
| Uscita: | nessuna |
| Registri modificati: | AF, BC, DE, HL, A'F', B'C', D'E' |
| Locazione: | X19FCH/1EF0H |

Viene aperto il canale logico 3 (usualmente il canale "P") e stampato il carattere.

SCAN_K: scandisce la matrice della tastiera

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 20H |
| Entrata: | nessuna |
| Uscita: | NZ se è stato premuto un tasto |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X1A01H/1EF5H |

Viene eseguita la scansione diretta della tastiera e se è stato premuto un qualunque tasto (inclusi quelli di SHIFT), il flag Z viene posto a 0 . Gli interrupt non sono necessari.

NEWVARS: crea le variabili di sistema aggiuntive

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 31H |

| | |
|---|---|
| Entrata: | nessuna |
| Uscita: | nessuna |
| Registri modificati: | AF, BC, DE, HL |
| Locazione: | X19A8H/1E98H |

Se le 58 variabili di sistema aggiuntive (in Appendice A) non sono già esistenti, vengono create, memoria permettendo. (In effetti la routine contiene solo l'istruzione RET e le variabili aggiuntive vengono create dall'istruzione RST 8).

SHADOW: chiama qualunque routine della ROM ombra

| | |
|---|---|
| Per l'uso: | RST 8<br>DEFB 32H |
| Entrata: | (HD_11)=locazione |
| Uscita: | nessuna |
| Registri modificati: | non specificabili (dipende dalla routine) |
| Locazione: | X19A4H/1E94H |

Viene chiamata la routine della ROM ombra puntata da HD_11. Ufficialmente riservato alla *Sinclair* è l'hook code più potente fra quelli presenti nella ROM. È possibile passare alla routine solo il valore del registro A, ma tutti i valori sono ricevibili. Può anche essere usato per disabilitare la ROM del BASIC sfruttando il seguente codice:

```
        LD HL,PAGOUT
        LD (HD_11),HL
        RST 8
        DEFB 32H
PAGOUT:POP HL              ;rimuove dallo stack 0700H
        POP HL             ;rimuove PAGOUT dallo stack
          "                ;resto del programma
          "
```

Se desiderate riabilitare la ROM del BASIC eseguite: CALL 0700H.

## Sommario degli hook code

Questo elenco riporta gli hook code, i relativi nomi, le locazioni nella ROM ombra e le funzioni.

| Codice | Nome | Locazione | Funzione |
|---|---|---|---|
| 1B | PAUSE | 19D9/1ECD | Attende che sia premuto un tasto — valore in A |
| 1C | PRINT | 19EC/1EE0 | Visualizza CHR$ A sul canale logico FE (lo schermo) |
| 1D | 232IN | 0B81/0B88 | RS232 input in A — carry se dato valido |
| 1E | 232OUT | 0C5A/0D07 | RS232 output — codice del carattere in A |
| 1F | LPRINT | 19FC/1EF0 | Stampa CHR$ A sul canale logico 3 (la stampante) |
| 20 | SCAN_K | 1A01/1EF5 | Scansione della matrice della tastiera — NZ se è stato premuto un tasto |
| 21 | MOTOR | 17F7/1532 | Accende e spegne i motori dei Microdrive |
| 22 | OPEN_M | 1B29/1B05 | Apre un canale "M" temporaneo |
| 23 | CLOSE_M | 12A9/138E | Chiude un canale "M" |
| 24 | ERASE_M | 1D6E/1D79 | Cancella un file da una cartuccia |
| 25 | READ_NP | 1A09/1EFD | Legge il prossimo buffer di PRINT |
| 26 | WRITE_S | 11FF/12DA | Manda il buffer "M" alla cartuccia |
| 27 | READ_P | 1A17/1F0B | Legge un buffer di PRINT |
| 28 | READ_R | 1A4B/1F3F | Legge random un file |
| 29 | READ_S | 1A86/1F7A | Legge serialmente un file |
| 2A | WRITE_R | 1A91/1F85 | Scrive random un blocco |
| *2B | MAKE_M | ----/10A5 | Crea un'area "M" |
| 2C | RECLAIM | 10C4/119F | Libera un'area "M" di 595 byte |
| 2D | OPEN_N | 0EA9/0F46 | Apre un canale "N" temporaneo |
| 2E | CLOSE_N | 1A24/1F18 | Chiude un canale "N" |
| *2F | READ_N | ----/1F25 | Legge un pacchetto dalla rete locale |
| 30 | WRITE_N | 0DB2/0E4F | Scrive un pacchetto sulla rete locale |
| 31 | NEWVARS | 19A8/1E98 | Crea, se necessario, le variabili aggiuntive di sistema |
| 32 | SHADOW | 19A4/1E94 | Chiama la routine della ROM ombra indirizzata da HD_11 |
| *33 | SCAN_M | ----/1FE4 | Legge il prossimo header dalla cartuccia |
| *34 | OPEN_B | ----/1FF6 | Apre il canale "B" |

*routine presenti solo nella nuova ROM.

# Note

1. Usate la funzione USR solo insieme ai comandi LET o RANDOMIZE. In caso contrario, il suo uso in comandi di BASIC esteso, in caso di errore, farà sì che la ROM ombra possa mandare in blocco il sistema.
2. Prestate attenzione allo stato degli interrupt in ingresso e in uscita da alcune routine della ROM ombra. Gli interrupt dovranno sempre essere abilitati prima di tornare al BASIC.
3. Salvate sempre il valore di H'L' se, dopo l'uso delle routine dei Microdrive desiderate tornare al BASIC.
4. Non usate mai cartucce importanti durante la correzione di routine dei Microdrive. Sarebbe estremamente spiacevole se accidentalmente formattaste l'unica cartuccia contenente tutto il vostro codice sorgente! La rimozione delle linguette di protezione contro la sovrascrittura non garantisce la sicurezza da possibili errori in codice macchina.
5. Assicuratevi di aver creato le variabili aggiuntive di sistema usando l'hook code 31H.
6. Non usate REM per le routine di interfaccia. La creazione delle aree CHANS li sposterà in su nella mappa della memoria durante l'esecuzione, causando il blocco della memoria.

Capitolo

# Comandi BASIC supplementari

# 9

Ai comandi di BASIC esteso forniti allo Spectrum dall'Interface 1, l'utente può aggiungere altri comandi di sua creazione. È questa una caratteristica molto utile, presente su molti altri computer, ma sempre assente nella gamma *Sinclair*. La principale ragione per la sua precedente mancanza è il fatto che, finora, tutti i comandi sullo Spectrum devono essere parole chiave e il set di caratteri non ha spazio per aggiungere comandi. L'Interface 1 permette di saltare a un opportuno programma in codice macchina in seguito ad un errore sintattico presente sulla linea, o al momento dell'introduzione, o al momento dell'esecuzione. Questo permette di cambiare la sintassi di molti comandi esistenti per alterarne la funzione, per quanto i comandi dell'Interface 1 non possano normalmente essere cambiati. Per comandi non-Interface 1, esistono vari modi per aggiungere funzioni:

1. Aggiungendo caratteri in strani punti: COPY #
2. Cambiando il tipo dei parametri: POKE 30000,"x"
3. Aggiungendo parametri: PLOT x, y, "BANG"
4. Omettendo parametri: CIRCLE 10,30
5. Usando le funzioni di modo E: LINE 10,30

Potete anche usare i vostri comandi, digitati interamente ma preceduti da un carattere non alfabetico, (ad esempio !REPEAT). Il carattere non alfabetico è necessario per fare uscire lo Spectrum dal modo "K", così da permettere la digitazione della linea. Le parole chiave, la cui sintassi non può normalmente essere modificata, sono LOAD, SAVE, MERGE, VERIFY, CAT, FORMAT, OPEN, CLOSE, CLS, CLEAR e MOVE. In effetti

è possibile effettuare modifiche, ma ciò comporta un po' di fatica. È possibile modificare i comandi facendoli precedere da un carattere "shiftato", ad esempio "*". Per quello che riguarda le parole chiave, questo implica la digitazione della parola chiave, lo spostamento del cursore a sinistra, la digitazione di "*", lo spostamento del cursore a destra e la digitazione del resto della linea.
Per creare i vostri comandi, dovete avere una buona conoscenza del codice macchina e del sistema operativo dello Spectrum: la seconda metà di questo capitolo descrive i metodi. Questa prima metà contiene un programma che altera la sintassi di consueti comandi per Microdrive e può essere tranquillamente digitato ed usato da qualunque utente.
Per informare l'Interface che sono stati aggiunti nuovi comandi, dovete eseguire due POKE, posti in una linea multicomando. Questa linea non deve essere digitata prima che sia stata eseguita almeno un'operazione di interfaccia dopo NEW. Il modo più facile è sfruttare CLOSE# che è in effetti un comando fittizio. Per far tornare l'interfaccia a riconoscere solo i comandi normali, dovrete eseguire

```
POKE 23735,240: POKE 23736,1
```

al solito in un'unica linea.
Devo ammettere che la sintassi scelta per il caricamento ed il salvataggio dei programmi su Microdrive non è entusiasmante. Dal mio punto di vista, essa è inutilmente complessa e richiede la pressione di troppi tasti. Ho pertanto scritto il seguente programma per ottenere comandi più semplici. Sono tutti sotto forma di un asterisco seguito dall'opportuna lettera di comando (in maiuscolo o in minuscolo) e da eventuali altri parametri. Per caricare un programma chiamato TEST dal drive 2, il nuovo comando sarà *L"2TEST", che è equivalente a LOAD *"m";2;"TEST". Analogamente, SAVE *"m", VERIFY e MERGE sono eseguiti anche da *S, *V, *M, seguiti da una singola stringa. Il primo carattere della stringa deve essere il numero del drive. Un altro comando aggiunto è *C, equivalente a CAT 1; *C seguito da un numero fornisce il catalogo degli altri drive. L'ultimo nuovo comando è *RUN, che carica un programma chiamato RUN dal drive 1, ed è equivalente a NEW seguito da RUN.

**Programma ESTENSIONE SINTASSI**

```
10 REM ************************
20 REM * Estensione sintassi *
30 REM ************************
40 REM                per 16K
50 CLEAR 65169:    REM 32401
60 LET ex=65170:  REM 32402
```

```
 65 LET x2=INT ((ex+124)/256): LET x1=ex+124-256*x2
 70 RESTORE 200: LET c=0
 80 FOR i=ex TO ex+194
 90 READ a: LET c=c+a
100 POKE i,a
110 NEXT i
115 IF c<>22003+2*(x1+x2) THEN  PRINT "Errore di chec
ksum": STOP
120 CLOSE #0
130 POKE 23735,ex-256*INT (ex/256): POKE 23736,INT (e
x/256)
140 PRINT "Nuovi comandi:"
150 PRINT "*L LOAD"'"*S SAVE"'"*M MERGE"'"*V VERIFY"
160 PRINT "*C CAT"'"*RUN"
170 STOP
200 DATA 254,92,194,240,1,215,32,0
210 DATA 246,32,254,115,40,22,254,108
220 DATA 40,28,254,118,40,34,254,109
230 DATA 40,36,254,99,40,38,254,114
240 DATA 40,62,24,222,253,203,124,238
250 DATA 205,x1,x2,195,54,8,253,203
260 DATA 124,230,205,x1,x2,195,165,8
270 DATA 253,203,124,254,24,244,253,203
280 DATA 124,246,24,238,33,214,92,54
290 DATA 1,35,54,0,35,54,2,215
300 DATA 32,0,254,13,40,7,254,58
310 DATA 40,3,205,30,6,195,169,4
320 DATA 215,32,0,246,32,254,117,194
330 DATA 40,0,215,32,0,246,32,254
340 DATA 110,32,244,215,32,0,205,183
350 DATA 5,195,149,10,62,77,50,217
360 DATA 92,215,32,0,215,140,28,215
370 DATA 24,0,223,202,35,7,215,241
380 DATA 43,33,11,0,167,237,66,218
390 DATA 76,6,120,177,202,76,6,26
400 DATA 214,48,254,1,56,27,254,9
410 DATA 48,23,50,214,92,175,50,215
420 DATA 92,11,19,237,67,218,92,237
430 DATA 83,220,92,215,24,0,195,35
440 DATA 7,231,4
```

Ho trovato questi nuovi comandi più semplici e veloci da usare ed ho salvato il programma con il nome RUN sulla mia principale cartuccia di sviluppo software. Introducendo questo programma, modificate le linee 50 e 60 per variare la RAMPTOP e la posizione di memoria dove andrà il programma in funzione della versione di Spectrum che avete. Le locazioni suggerite sono incompatibili con quelle consigliate per alcune altre routine di questo libro. Se volete che siano tutte compatibili fra loro, aggiungete le linee:

```
50 CLEAR 64579: REM 31809 per 16K
60 LET mc=64580: REM 31810 per 16K
```

Ogni comando aggiuntivo sarà disabilitato dopo un NEW. Se desiderate quindi cancellare un programma in memoria, senza eliminare i comandi extra, dovrete creare un file di programma vuoto su cartuccia, chiamato NEW e caricarlo con

```
*L"1NEW"
```

## Come aggiungere comandi

Questa parte del capitolo è dedicata ai programmatori in codice macchina che abbiano una buona conoscenza del sistema operativo dello Spectrum.
Fondamentale per la possibilità di aggiungere comandi, è la variabile di interfaccia VECTOR, in 5CB7H. Questa normalmente contiene 01F0H, che è la locazione della ROM ombra che esegue la normale routine di gestione di errore (dopo aver copiato CHADD_ in CH_ADD).
Per spiegare come alterare VECTOR, è necessario esaminare l'azione della ROM ombra in seguito ad un errore. I numeri tra parentesi si riferiscono alle locazioni della ROM espresse in esadecimale.
Innanzitutto in HL viene posto l'indirizzo di ritorno (0013) (da RST 8), che viene controllato per vedere se sia 0000 o 15FE. Se è 0000, allora la ROM ombra ha chiamato una routine della ROM 16K; se invece è 15FE, è stato richiesto un canale di interfaccia. Se non è nessuno di questi due, vengono create (nel caso non esistano già) le opportune variabili di interfaccia e vengono emessi gli opportuni segnali alla ULA. Viene poi trovato il numero di errore (00C7) (dal byte seguente RST 8), che viene controllato per vedere se si tratti di un byte di hook code. In caso negativo, il byte viene controllato per vedere se l'errore fosse NONSENSE IN BASIC, INVALID FILENAME o INVALID STREAM. Se non è nessuno di questi viene usata la routine di gestione degli errori presente nella vecchia ROM (00F8). Se era invece uno degli errori sopra menzionati, il contenuto di CH_ADD viene immagazzinato in CHADD_ (che confusione di mnemonici!). Viene controllato ora il bit 3 di FLAGS3: se esso è ad 1, viene usata la normale gestione degli errori. Dopo aver calcolato il primo carattere nella linea in cui è avvenuto l'errore (0126–01E9), esso viene confrontato con i token dell'Interface (01B5–01E9). Se non è riconosciuto, allora si salta alla routine puntata da VECTOR. Normalmente questo produce un errore ma, se VECTOR viene cambiato, potete aggiungere comandi che non passerebbero i normali controlli sintattici.

Pertanto, se avete cambiato VECTOR per farlo puntare alla vostra routine, cosa dovrà fare questa? Dovrà innanzitutto controllare il contenuto del registro A, per vedere se sia il primo token o codice di carattere di uno dei vostri comandi extra, meno 206; se esso non è riconosciuto, allora JP 01F0H per produrre un errore. In caso di riconoscimento, dovrete cedere il controllo alla routine di comando, che controllerà la sintassi, valuterà i parametri ed eseguirà l'istruzione (le ultime due operazioni vengono eseguite solo durante l'esecuzione del programma). È bene essere al corrente di alcune cose, per poter scrivere una routine idonea:

1. È selezionata la ROM ombra, non quella del BASIC
2. Tutti i restart Z80 sono differenti
3. Non tentate di usare gli hook code, chiamate le routine direttamente
4. Anche se gli interrupt sono abilitati, non viene eseguita la scansione della tastiera e non viene incrementato FRAMES

Le restart ombra eseguono le seguenti operazioni:

RST 0 — Resetta FLAGS3 e cede il controllo alla ROM 16K
RST 8 — Da non usare
RST 10 — Chiama la routine della ROM 16K — va seguito da 2 data byte che indichino la locazione
RST 19 — BIT 7, (FLAGS) — Z se controllo sintattico, NZ se esecuzione.
RST 20 — Genera un messaggio di errore sulla ROM ombra. Va seguito da un byte di dati:

| | | |
|---|---|---|
| FF | PROGRAM FINISHED | 00E7 |
| 00 | NONSENSE IN BASIC | 0139 |
| 01 | INVALID STREAM NUMBER | 0663 |
| 02 | INVALID DEVICE EXPRESSION | 062D |
| 03 | INVALID NAME | 064C |
| 04 | INVALID DRIVE NUMBER | 0681 |
| 05 | INVALID STATION NUMBER | 05F6 |
| 06 | MISSING NAME | 068D |
| 07 | MISSING STATION NUMBER | 06A1 |
| 08 | MISSING DRIVE NUMBER | 0683 |
| 09 | MISSING BAUD RATE | 06B7 |
| 0A | HEADER MISMATCH ERROR | (mai usato nella ROM) |
| 0B | STREAM ALREADY OPEN | 052F |
| 0C | WRITING TO A 'READ' FILE | 0D78 |
| 0D | READING A 'WRITE' FILE | 0D1C |
| 0E | DRIVE 'WRITE' PROTECTED | 128D |
| 0F | MICRODRIVE FULL | 1219 |

| | | |
|---|---|---|
| 10 | MICRODRIVE NOT PRESENT | 1828 |
| 11 | FILE NOT FOUND | 11A3 |
| 12 | HOOK CODE ERROR | 1985 |
| 13 | CODE ERROR | 092E |
| 14 | MERGE ERROR | 07D8 |
| 15 | VERIFICATION HAS FAILED | 0930 |
| 16 | WRONG FILE TYPE | 0902 |

(Le locazioni esadecimali su riportate sono gli indirizzi dei salti che producono ogni singolo messaggio di errore).

RST 28 — Genera un errore sulla ROM 16K. Prima di eseguire RST, ponete l'opportuno codice di errore in ERRNR
RST 30 — Crea, se non esistenti, le variabili di interfaccia
RST 38 — Abilita gli interrupt (questo è il motivo per cui non viene eseguita la scansione della tastiera)

## Routine di scansione della linea

Qualunque comando aggiungiate, dovrete esaminare la linea BASIC per la sintassi o l'esecuzione. La vostra routine di comando sarà chiamata due volte per ogni linea inserita: una per il controllo sintattico e l'altra per l'esecuzione. Per conoscere lo stato, usate RST 18. La vecchia ROM contiene varie routine di scansione della linea che potete sfruttare usando RST 10. Per esaminare la linea carattere per carattere, usate le seguenti routine della vecchia ROM:

GET_CHAR 0018H — Il registro A contiene il byte corrente della linea
NEXT_CHAR 0020H — Il registro A contiene il successivo byte della linea, ignorando i codici di controllo e di spazio

Le principali routine di valutazione sono:

CLASS_061C82H — Controlla/valuta i parametri numerici, ponendone il valore sullo stack di calcolo, o inserendo byte invisibili nella linea
CLASS_0A1C8CH — Controlla/valuta i parametri delle stringhe ponendo i valori sullo stack di calcolo durante l'esecuzione

Prima di cedere il controllo alle ultime due routine descritte, CH_ADD deve puntare al primo carattere dell'espressione. Al ritorno, esso punterà al primo carattere "fuori posto", di cui A conterrà il codice (ad esempio ",").
Anche la ROM ombra possiede alcune routine di scansione che possono essere chiamate direttamente.

| | |
|---|---|
| PARAMS #0701 | — La routine controlla *stringa; numero <;stringa: parametri>*, cioè i caratteri che seguono LOAD, SAVE ... Tutti i parametri vengono passati, al momento dell'esecuzione, a D_STR1, N_STR1, S_STR1, L_STR1, e HD_00 — HD_11 |
| EVALBC #061E | — Valuta un numero a 16 bit, che viene restituito in BC e in D_STR1 |
| CHKEND #05B7 | — Controlla se fine di comando. Se no, viene generato un errore; se si è raggiunta la fine, viene generato un RET durante l'esecuzione, altrimenti viene rimosso l'indirizzo di ritorno e il controllo viene ceduto alla ROM 16K attraverso 05C1 |
| CHKDRV 066D | — Controlla che il contenuto di D_STR1 sia nel campo 1 – 8. Se no, viene generato l'errore INVALID DRIVE NUMBER |
| CHKST$ #062F | — Valuta una stringa. Durante l'esecuzione viene controllato che il numero di caratteri sia inferiore ad 11, i parametri vengono immagazzinati in N_STR1 e T_STR1 |

Eseguito il controllo sintattico e verificata la sua correttezza, va eseguito un salto a 05C1H (salvo che non venga fatto un ritorno indiretto attraverso CHKEND). Quando un comando è stato eseguito con successo, va eseguito un salto simile. Viene azzerato il codice di errore e si ritorna alla ROM 16K. Come si può vedere, aggiungere comandi è un po' difficile, ma ne vale la pena. Viene così messa a disposizione dello Spectrum una vastissima gamma di sussidi di programmazione e si ha anche la possibilità di usare l'editor di linea del BASIC per scrivere programmi in altri linguaggi o in Assembler.
Presentiamo ora il listato in Assembler dei comandi aggiunti nella prima metà di questo capitolo. La sua posizione è obbligata: il caricatore BASIC altera i byte dei CALL automaticamente via via che riloca.

```
0010;                                 Modifica sintassi
0030            ORG     40000             origine di debug
0040            CP      5CH               controlla se "*"-206
```

```
0050 ERROR   JP    NZ,01F0H     errore se no
0060         RST   10H
0070         DEFW  0020H        call NEXT_CHAR
0080         OR    20H          trasforma in minuscolo
0090         CP    "s"
0100         JR    Z,SAVE       salta se "s" o "S"
0110         CP    "l"
0120         JR    Z,LOAD       salta se "l" o "L"
0130         CP    "v"
0140         JR    Z,VERIF      salta se "v" o "V"
0150         CP    "m"
0160         JR    Z,MERGE      salta se "m" o "M"
0170         CP    "c"
0180         JR    Z,CAT        salta se "c" o "C"
0190         CP    "r"
0200         JR    Z,RUN        salta se "r" o "R"
0210         JR    ERROR        errore in caso contrario
0220 SAVE    SET   5,(IY+124)   segnala SAVE
0230         CALL  VARS         valuta la stringa
0240         JP    0836H        esegue il SAVE
0250 LOAD    SET   4,(IY+124)   segnala LOAD
0260 DOIT    CALL  VARS         valuta la stringa
0270         JP    08A5H        esegue LOAD/VERIFY/MERGE
0280 VERIF   SET   7,(IY+124)   segnala VERIFY
0290         JR    DOIT         lo esegue
0300 MERGE   SET   6,(IY+124)   segnala MERGE
0310         JR    DOIT         lo esegue
0320 CAT     LD    HL,5CD6H     HL = D_STR1
0330         LD    (HL),1       drive 1
0340         INC   HL
0350         LD    (HL),0       azzera il byte più significativo
0360         INC   HL
0370         LD    (HL),2       canale logico 2
0380         RST   10H
0390         DEFW  0020H        call NEXT_CHAR
0400         CP    13
0410         JR    Z,CAT2       salta se "a capo"
0420         CP    ":"
0430         JR    Z,CAT2       salta se separatore
0440         CALL  061EH        valuta un numero
0450 CAT2    JP    04A9H        esegue il CAT
0460 RUN     RST   10H
```

```
0470          DEFW  0020H        call next-char
0480          OR    20H          esegue il CAT
0490          CP    "u"
0500 ERR2     JP    NZ,0028H     errore se non "u"
0510          RST   10H
0520          DEFW  0020H        call NEXT_CHAR
0530          OR    20H
0540          CP    "n"
0550          JR    NZ,ERR2      errore se non "n"
0560          RST   10H
0570          DEFW  0020H        call NEXT_CHAR
0580          CALL  05B7H        call CHKEND
0590          JP    0A95H        carica il programma "run"
0600 VARS     LD    A,"M"
0610          LD    (5CD9H),A    pone L_STR1 = "M"
0620          RST   10H
0630          DEFW  0020H        call NEXT_CHAR
0640          RST   10H
0650          DEFW  1C8CH        call CLASS_0A (stringa)
0660          RST   10H
0670          DEFW  0018H        call GET_CHAR
0680          RST   18H          controlla la sintassi
0690          JP    Z,0723H      salta se solo controllo sin-
                                 tattico
0700          RST   10H
0710          DEFW  2BF1H        call STK_FETCH (BC =
0720          LD    HL,11        lunghezza, DE = inizio)
0730          AND   A
0740          SBC   HL,BC
0750          JP    C,064CH      errore se più di 11 caratteri
0760          LD    A,B
0770          OR    C
0780          JP    Z,064CH      errore se stringa vuota
0790          LD    A,(DE)       A = codice del primo ca-
                                 rattere
0800          SUB   "0"          A = numero drive
0810          CP    1
0820          JR    C,INVDR      errore se <1
0830          CP    9
0840          JR    NC,INVDR     errore se >8
0850          LD    (5CD6H),A    immagazzina il numero in
                                 D_STR1
0860          XOR   A
```

| | | | | |
|---|---|---|---|---|
| 0870 | | LD | (5CD7H),A | azzera il byte più significativo di D_STR1 |
| 0880 | | DEC | BC | decrementa la lunghezza |
| 0890 | | INC | DE | incrementa l'indirizzo iniziale |
| 0900 | | LD | (5CDAH), BC | immagazzina la lunghezza |
| 0910 | | LD | (5CDCH), DE | immagazzina l'indirizzo iniziale |
| 0920 | | RST | 10H | |
| 0930 | | DEFW | 0018H | call GET_CHAR |
| 0940 | | JP | 0723H | controlla gli argomenti, quindi RET |
| 0950 | INVDR | RST | 20H | errore |
| 0960 | | DEFB | 4 | byte per "INVALID DRIVE NUMBER" |

Come vedete, ho sfruttato alcune routine della ROM ombra abbastanza oscure. Queste locazioni non vanno chiamate usando l'hook code 32H, poiché sfruttano le routine di scansione di linea e ritornano al BASIC e non al codice macchina chiamante.

**Appendice**

# Le variabili di sistema dell'interfaccia A

I seguenti 58 byte vengono aggiunti alla mappa della memoria:

| Decimale | Esadec. | Nome | Uso |
|---|---|---|---|
| 23734 | 5CB6 | FLAGS3 | bit 0 — posto ad 1 durante l'uso dei comandi BASIC esteso<br>bit 1 — posto ad 1 durante CLEAR #<br>bit 2 — posto ad 1 se ERRSP alterato<br>bit 3 — posto ad 1 durante le operazioni in rete locale<br>bit 4 — posto ad 1 durante LOAD<br>bit 5 — posto ad 1 durante SAVE<br>bit 6 — posto ad 1 durante MERGE<br>bit 7 — posto ad 1 durante VERIFY |
| 23735 | 5CB7 | VECTOR | Usato per estendere l'interprete — normalmente 01F0 |
| 23737 | 5CB9 | SBRT | Routine usata dalla ROM ombra per chiamare le routine della ROM 16K, del tipo:<br>LD HL valore<br>CALL routine<br>LD (5CBAH), HL<br>RET |

| Decimale | Esadec. | Nome | Uso |
|---|---|---|---|
| 23747 | 5CC3 | BAUD | Velocità RS232 (baud rate) — inizialmente 000CH, 19200 — pari a circa (3500000/(26*rate))−2 |
| 23749 | 5CC5 | NTSTAT | Numero della stazione durante le operazioni in rete locale (1−64) |
| 23750 | 5CC6 | IOBORD | Colore della cornice durante le operazioni I/O |
| 23751 | 5CC7 | SER_FL | Spazio di lavoro RS232 — il primo byte è un flag, il secondo è un carattere in input |
| 23753 | 5CC9 | SECTOR | Spazio di lavoro del Microdrive — normalmente usato per contare i settori, iniziando da FFH o 04FBH |
| 23755 | 5CCB | CHADD_ | Immagazzinamento temporaneo per CH_ADD durante il controllo delle linee di sintassi estesa |
| 23757 | 5CCD | NTRESP | Codice di risposta della rete locale — numero della stazione ricevente — il consenso iniziale per l'intestazione di rete locale ad 8 byte |
| 23758 | 5CCE | NTDEST | Numero della stazione di rete locale destinataria |
| 23759 | 5CCF | NTSRCE | Numero della stazione di rete locale sorgente |
| 23760 | 5CD0 | NTNUMB | Numero di blocco della rete locale (0−65535) |
| 23762 | 5CD2 | NTTYPE | Codice di tipo dell'intestazione della rete locale — 0 dati, 1 EOF |
| 23763 | 5CD3 | NTLEN | Lunghezza blocco dati della rete locale (0−255) |
| 23764 | 5CD4 | NTDCS | Checksum del blocco dati della rete locale |
| 23765 | 5CD5 | NTHCS | Checksum dell'intestazione della rete locale (di NTDEST−NTDCS) |
| 23766 | 5CD6 | D_STR1 | Inizio del primo specificatore di file ad 8 byte: numero di drive (1−8) o numero di stazione destinatario in |

| Decimale | Esadec. | Nome | Uso |
|---|---|---|---|
| | | | operazione in rete locale o velocità di trasferimento |
| 23768 | 5CD8 | S_STR1 | Numero di canale logico (0 – 15) |
| 23769 | 5CD9 | L_STR1 | Identificatore di canale (maiuscolo) |
| 23770 | 5CDA | N_STR1 | Lunghezza del nome del file |
| 23772 | 5CDC | T_STR1 | Inizio del nome del file (normalmente nell'area di lavoro) |
| 23774 | 5CDE | D_STR2 | Inizio del secondo specificatore di file ad 8 byte |
| 23782 | 5CE6 | HD_00 | Tipo file: 0 — programma<br>1 — array numerico<br>2 — array stringa<br>3 — byte |
| 23783 | 5CE7 | HD_0B | Lunghezza dei dati |
| 23785 | 5CE9 | HD_0D | Inizio dei dati |
| 23787 | 5CEB | HD_0F | Lunghezza del programma (o nome dell'array) |
| 23789 | 5CED | HD_11 | Numero di linea per l'autoesecuzione (usato anche dall'hook code 32H) |
| 23791 | 5CEF | COPIES | Numero di copie multiple eseguite da SAVE — viene reinizializzato ad 1 dopo il SAVE |

Note: FLAGS3 viene normalmente posto a 0 ogni volta che la ROM ombra non è selezionata. Può essere utilmente indirizzato da IY+124.
I parametri nelle routine SBRT vengono posti in memoria (POKE) dalla ROM ombra.
L'istruzione RETURN va alla locazione 8, ma sullo stack è sempre presente anche 0000 per differenziarla da un errore.
I nomi di variabile da HD_00 a HD_11 vengono prelevati direttamente dai loro equivalenti usati dalle routine di gestione cassette della ROM 16K indirizzati da IX+0 a IX+11H.

**Appendice B**

# Listati in Assembler

Questa appendice contiene i listati in Assembler Z80 delle routine usate in questo libro. Nei listati i numeri esadecimali sono seguiti dalla lettera H; i numeri di linea sono usati dall'Assembler e non hanno qui alcun significato. Opportuni commenti sono stati aggiunti all'estrema destra. Tutte le ORG sono arbitrarie, poiché la gran parte delle routine è o indipendente dalla posizione o autorilocante. Quelle che sono autorilocanti usano la caratteristica che alla loro chiamata il registro BC contiene il loro indirizzo iniziale.

## CANALE LOGICO 14-z$

Questa routine crea un nuovo canale "Z" e lo abbina al canale logico 14, che si suppone sia stato precedentemente chiuso. Si ritiene anche che siano già state create le 58 variabili di sistema di interfaccia. Questo è il motivo per cui il comando CLOSE # viene incluso nel caricatore BASIC, che fa tutto il necessario.

```
;                      Routine per inserire nella
;                      stringa z$ i caratteri in uscita
;                      dal canale logico #14
;                      "(copyright) A. Pennell 1983"
PROG      EQU          5C53H
VARS      EQU          5C4BH
          ORG          40000
SETUP     LD           HL,(PROG)
```

```
0100          DEC    HL              HL = PROG-1 = fine di
                                     CHANS
0110          PUSH   BC              salva l'indirizzo iniziale
0120          PUSH   HL              salva l'indirizzo iniziale della nuo-
                                     va area dati
0130          LD     BC,11           servono 11 byte
0140          CALL   1655H           call MAKE_ROOM
0150          POP    DE              DE = indirizzo iniziale della nuo-
                                     va area dati
0160          LD     HL,OUTCH-       SETUP
0170          POP    BC              riprende SETUP
0180          ADD    HL,BC           HL = OUTCH
0200          PUSH   DE              salva l'indirizzo iniziale dei dati
0210          EX     DE,HL
0220          LD     (HL),E          salva OUTCH nella nuova area
0230          INC    HL
0240          LD     (HL),D
0250          INC    HL
0260          EX     DE,HL
0270          LD     BC,LABEL-       OUTCH
0280          ADD    HL,BC           HL = LABEL
0290          LD     BC,9            altri 9 byte di dati
0300          LDIR                   copia i dati in CHANS
0310          POP    HL              HL = indirizzo iniziale dei dati
0320          INC    HL
0330          LD     BC,(5C4FH)      BC = CHANS
0340          AND    A
0350          SBC    HL,BC
0360          LD     (5C32H),HL      salva lo spiazzamento in STRMS
                                     per #14
0370          LD     BC,0
0380          RET                    ritorna al BASIC con BC = 0
                                     i rimanenti dati vanno in
                                     CHANS:
0390 LABEL    DEFW   15C4H           routine "input"
0400          DEFM   "Z"             identificatore di canale
0410          DEFW   28H             ROM ombra output
0420          DEFW   28H             ROM ombra input
0430          DEFW   11              numero di byte
                                     routine di output:
0440 OUTCH    PUSH   AF              salva il codice del carattere
0450          LD     HL,(VARS)
0460 L1       LD     A,(HL)
0470          CP     5AH
```

```
0480            JR      Z,FOUND     salta se ha trovato Z$
0490            CP      80H
0500            JP      Z,0670H     VARIABLE NOT FOUND
0510            CALL    19B8H       call NEXT_ONE
0520            EX      DE,HL       HL = inizio della prossima varia-
                                    bile
0530            JR      L1          torna in loop
0540 FOUND      INC     HL          HL = byte d'ordine basso della
                                    lunghezza della stringa
0550            LD      C,(HL)
0560            INC     HL          HL = byte d'ordine alto della lun-
                                    ghezza della stringa
0570            LD      B,(HL)      BC = lunghezza stringa
0580            INC     BC          incrementa la lunghezza
0590            PUSH    BC          la salva
0600            PUSH    HL          salva l'indirizzo del primo carat-
                                    tere
0610            ADD     HL,BC       HL = indirizzo finale della stringa
0620            CALL    1652H       call ONE_SPACE (fa posto al ca-
                                    rattere)
0630            INC     HL          HL = nuovo spazio
0640            EX      DE,HL       DE = nuovo spazio
0650            POP     HL          HL = indirizzo del primo carattere
0660            POP     BC          BC = nuova lunghezza
0670            LD      (HL),B      immagazzina la nuova
                                    lunghezza
0680            DEC     HL
0690            LD      (HL),C
0700            POP     AF          riprende il codice del carattere
0710            LD      (DE),A      lo aggiunge alla stringa
0720            AND     A           carry = 0 (per evitare errori)
0730            RET                 torna al sistema operativo
0740                    END
```

## ON EOF GO TO

Questa routine altera ERRSP per farlo puntare ad un'opportuna routine di gestione di errore.

```
0010;                   ON EOF GO TO
0020;                   "PIC code"
0040            ORG     40000
0050 SETUP      LD      HL,START-   SETUP altera ERRSP:
```

```
0060        ADD   HL,BC          HL = START
0070        EX    DE,HL          DE = START
0080        LD    HL,(ERRSP)     HL = ERRSP
0090        LD    (HL),E         immagazzina nello stack d'erro-
0100        INC   HL             re la nuova routine di errore
0110        LD    (HL),D
0120        RST   8              accerta la presenza delle varia-
0130        DEFB  31H            bili dell'Inferface
0140        LD    BC,0
0150        RET                  torna al BASIC con BC = 0
                                 gestore di errori:
0160 START  LD    HL,(ERRSP)     HL = locazione dello stack
0170        LD    A,(5C3AH)      A = codice di errore
0180        CP    EOF            è un EOF?
0190        JP    NZ,1303H       no, salta al gestore in ROM
0200        LD    E,(HL)         sì, DE = START
0210        INC   HL
0220        LD    D,(HL)
0230        PUSH  DE             appende START allo stack
0240        CALL  16B0H          pulisce l'area di lavoro e l'area di
                                 edit
0250        RES   5,(IY = 37H)   segnala "pronto per un nuovo ta-
                                 sto"
0260        CALL  0D6EH          svuola la parte inferiore dello
                                 schermo e apre il canale logico 0
0270        LD    HL,(5C45H)     HL = numero della linea corrente
0280        LD    (5CC9H),HL     lo salva in SECTOR
0290        LD    DE,LINE        DE = linea cui saltare
0300        LD    HL,5C42H       HL = NEWPPC
0310        LD    (HL),E         salva il nuovo numero di linea
0320        INC   HL
0330        LD    (HL),D
0340        INC   HL
0350        LD    (HL),1         pone NSPPC = 1 e torna alla
0370        JP    1B7DH          ROM
0380 ERRSP  EQU   5C3DH
0390 EOF    EQU   7              codice d'errore di EOF
0400 LINE   EQU   1000           numero della linea cui saltare
0410        END                  in caso d'errore
```

**OPEN # OGNI TIPO**

Questa routine apre un determinato canale logico su un canale Microdrive, senza operare alcuna restrizione sul tipo di file. È in pratica la routine OPEN # con una modifica al controllo di errore.

```
0010;                                  "OPEN # ogni tipo"
0030;
0040;
0050 OPENM   EQU    22H            codici di interfaccia
0060 CLOSM   EQU    23H
0070 MOTOR   EQU    21H
0080         ORG    40000
0090 START   LD     A,(5CD8H)      A = S_STR1 = numero di canale
                                   logico
0100         CALL   1727H          call STR_DATA1
0110         LD     HL,17
0120         XOR    A
0130         SBC    HL,BC
0140         LD     BC,0
0150         RET    C              ritorna al BASIC con BC = 0 se il
                                   canale è già aperto
0160         LD     (5CD7H),A      azzera il byte d'ordine alto di
                                   D_STR1
0170         LD     HL,10
0180         LD     (5CDAH),HL     lunghezza del nome del file = 10
0190         LD     HL,(5C7BH)
0200         LD     (5CDCH),HL     inizio del nome del file = area
                                   UDG
0210         LD     A,(5CD8H)      A = numero del canale logico
0220         ADD    A              calcola il relativo STRM
0230         LD     HL,5C16H
0240         LD     E,A
0250         LD     D,0
0260         ADD    HL,DE          HL = locazione canale logico
0270         EXX
0280         PUSH   HL             salva H'L'
0290         EXX
0300         PUSH   HL             salva la locazione STRM
0310         RST    8              apre un canale temporaneo "M"
0320         DEFB   OPENM
0330         BIT    0,(IX+24)
0340         JR     Z,READ         salta se è un file di lettura
0350         XOR    A              file non trovato, quindi
```

```
0360        RST    8           spegne tutti i motori
0370        DEFB   MOTOR
0380        RST    8           e libera l'area allocata
0390        DEFB   2CH
0400        POP    HL          ripristina lo stack
0410        EXX
0420        POP    HL          riprende H'L'
0430        EXX
0440        LD     BC,1
0450        RET                ritorna con BC = 1 se il file non è
                               stato trovato
0460 READ   RES    7,(IX+4)    rende permanente il canale "M"
0470        XOR    A
0480        PUSH   HL          salva lo spiazzamento del canale
                               logico
0490        RST    8           spegne i motori
0500        DEFB   MOTOR
0510        POP    DE          DE = spiazzamento
0520        POP    HL          HL = locazione
0530        LD     (HL),E      salva il nuovo spiazzamento
0540        INC    HL          nell'area STRMS
0550        LD     (HL),D
0560        LD     A,(IX+67)   A = RECFLG
0570        AND    4           maschera il bit che identifica i fi-
                               le di tipo PRINT
0580        ADD    2           aggiunge 2
0590        LD     C,A         salva il nuovo RECFLG in BC
0600        LD     B,0
0610        EXX
0620        POP    HL          riprende H'L'
0630        EXX
0640        RET                torna con BC = 2 se file di tipo
                               PRINT, con BC = 6 in caso con-
                               trario
0650        END
```

**STATO**

Questa routine è simile alla routine MOTOR nella ROM ombra, dalle locazioni X182AH fino a X1871H, con alcune modifiche nella gestione degli errori. Il suo scopo è di determinare se un particolare Microdrive è presente e se la cartuccia contenuta è protetta dalla scrittura.
Le label sono basate sulle equivalenti locazioni della ROM ombra. L'ORG

è un'origine di debug — il codice non è indipendente dalla posizione — il caricatore BASIC provvede alla rilocazione delle quattro CALL.

```
0010;                  STATO        routine
0020 MOTOR   EQU       21H          codice d'interfaccia per il control-
                                    lo motori
0030         ORG       40000        origine di debug
0040         LD        A,(5CD6H)    A = D_STR1 = numero drive
0050         DI                     disabilita gli interrupt
0060         JR        X182A        salta a X182A
0070 X1806   LD        HL,1388H     HL = 5000
0080 X1809   DEC       HL
0090         LD        A,L
0100         OR        H
0110         JR        NZ,X1809     attende un po'
0120         LD        HL,1388H
0130 X1811   LD        B,6
0140 X1813   IN        A,(0EFH)     legge dal port EF
0150         AND       4            esamina solo il bit 2
0160         JR        NZ,X1820     salta se il drive non è presente
0170         DJNZ      X1813        la presenza del drive viene con-
                                    trollata 6 volte
0180         JR        PRESN        se il drive è presente, salta
0190 X1820   DEC       HL           decrementa il contatore
0200         LD        A,H
0210         OR        L
0220         JR        NZ,X1811     riprova 5000 volte
0230         LD        BC,0         il drive non può essere connesso,
0240         JR        NOTPR        quindi ritorna con BC = 0
0250 X182A   LD        DE,0100H     D = 1, E = 0
0260         NEG                    complementa il numero del drive
0270         ADD       9            A = 9 - numero del drive
0280         LD        C,A          C = drive selezionato
0290         LD        B,8          B = contatore di drive
0300 X1835   DEC       C            decrementa il drive selezionato
0310         JR        NZ,X184B     salta se non è il drive sotto esame
0320         LD        A,D
0330         LD        (0F7H),A     invia 1 alla porta F7 - ON
0340         LD        A,0EEH
0350         OUT       (0EFH),A     invia EE alla porta EF
0360         CALL      X1867        attende un po'
0370         LD        A,0ECH
0380         OUT       (0EFH),A     invia EC alla porta EF
0390         CALL      X1867        attende un po'
```

```
0400        JR     X185C      passa al successivo
                              spegne un drive
0410 X184B  LD     A,0EFH
0420        OUT    (0EFH),A   invia EF alla porta EF
0430        LD     A,E
0440        OUT    (0F7H),A   invia 0 alla porta F7 – OFF
0450        CALL   X1867      attende un po'
0460        LD     A,0EDH
0470        OUT    (0EFH),A   invia ED alla porta EF
0480        CALL   X1867      attende un po'
0490 X185C  DJNZ   X1835      ripete per 8 drive
0500        LD     A,D
0510        OUT    (0F7H),A   invia 1 alla porta F7
0520        LD     A,0EEH
0530        OUT    (0EFH),A   invia EE alla porta EF
0540        JR     X1806      ne controlla lo stato

0550 X1867  PUSH   BC         routine di ritardo
0560        PUSH   AF         salva i registri
0570        LD     BC,0087H   BC = 135
0580 X18FB  DEC    BC
0590        LD     A,B
0600        OR     C
0610        JR     NZ,X18FB   ripete il ciclo 135 volte
0620        POP    AF         riprende i registri
0630        POP    BC
0640        RET               return
0650 PRESN  IN     A,(0EFH)   il drive è presente, quindi
0660        AND    1          controlla se sia stata rimossa la
                              linguetta di protezione
0670        LD     BC,1
0680        JR     NZ,NOTPR   salta con BC = 1 se la linguetta è
                              presente
0690        INC    BC         BC = 2 in caso contrario
0700 NOTPR  PUSH   BC         salva il parametro di ritorno
0710        XOR    A
0720        RST    8
0730        DEFB   MOTOR      spegne tutti i motori
0740        POP    BC         riprende il parametro
0750        RET               torna al BASIC
0760        END
```

## ON ERROR GO TO

Questa routine altera ERR_SP per farlo puntare a una nuova routine di gestione degli errori. Questo da solo non è sufficiente per gli errori di interfaccia, per cui il bit 2 di FLAGS3 deve essere posto a 1, in caso contrario verrà usata la normale routine di gestione degli errori. Se l'errore è capitato mentre era selezionata la ROM ombra, la coppia di registri HL conterrà 81H.

```
0010;         ON ERR GO TO
0020;         versione "Interface"
0030 ERRSP    EQU    5C3DH       predispone le costanti
0040 SECTR    EQU    5CC9H
0050 LINE     EQU    1000        linea cui saltare in caso di errore
0060          ORG    40000       origine arbitraria
0070 SETUP    LD     HL,START-SETUP
0080          ADD    HL,BC       HL = START
0090          EX     DE,HL       DE = START
0100          LD     HL,(ERRSP)  HL = errore di posizione nello
                                 stack
0110          LD     (HL),E      mette sullo stack la nuova routine
0120          INC    HL          di gestione
0130          LD     (HL),D
0140          RST    8
0150          DEFB   31H         crea le variabili aggiuntive di si-
                                 stema
0160          LD     BC,0
0170          RET                ritorna con BC = 0
                                 la nuova routine di gestione degli
                                 errori
0180 START    LD     (SECTR),HL  salva HL
0190          LD     HL,(ERRSP)
0200          LD     DE,1303H    DE = vecchia routine ROM di ge-
                                 stione errori
0210          PUSH   DE          salvando DE sullo stack la routi-
                                 ne è provvisoriamente disabilitata
0220          CALL   16B0H       pulisce alcune aree di memoria
0230          RES    5,(IY+37H)  resetta l'opportuno bit di FLAGS
0240          CALL   0D6EH       svuota la parte inferiore dello
                                 schermo e apre il canale logico 0
0250          LD     HL,17B9H    HL = routine ombra che libera i
                                 canali temporanei e spegne i mo-
                                 tori
0260          LD     (5CEDH),HL  pone HL in HD_11
```

| | | | | |
|---|---|---|---|---|
| 0270 | | LD | A,(5C3AH) | A = numero d'errore (vecchio errore) |
| 0280 | | LD | (23763),A | lo pone in NTLEN |
| 0290 | | PUSH | AF | salva il codice di errore |
| 0300 | | RST | 8 | |
| 0310 | | DEFB | 32H | call X17B9H |
| 0320 | | LD | HL,0081H | |
| 0330 | | LD | DE,(SECTR) | DE = valore di HL dopo l'errore |
| 0340 | | POP | AF | A = codice di errore della vecchia |
| 0350 | | AND | A | ROM |
| 0360 | | SBC | HL,DE | |
| 0370 | | JR | NZ,OLD | salta se all'ingresso HL< >81H cioè se errore gestito dalla vecchia ROM |
| 0380 | | BIT | 0,(IY+124) | controlla FLAGS3 |
| 0390 | | JR | NZ,OLD | se l'errore è in mezzo a un comando, viene gestito dalla vecchia ROM |
| 0400 | | CP | 7 | |
| 0410 | | JR | Z,OLD | salta se EOF |
| 0420 | | LD | A,100 | errore di interfaccia, quindi pone |
| 0430 | | LD | (23763),A | 100 in NTLEN |
| 0440 | | LD | A,"?" | |
| 0450 | | RST | 10H | visualizza un "?" |
| 0460 | | JR | PRNTN | visualizza i numeri di linea ecc. errore gestito dalla vecchia ROM |
| 0470 | | INC | A | |
| 0480 | | LD | B,A | numero di errore in B |
| 0490 | | CP | 10 | |
| 0500 | | JR | C,2 | |
| 0510 | | ADD | 7 | forma il carattere di errore |
| 0520 | | CALL | 15EFH | lo visualizza |
| 0530 | | LD | A," " | |
| 0540 | | RST | 10H | visualizza uno spazio |
| 0550 | | LD | A,B | |
| 0560 | | LD | DE,1391H | |
| 0570 | | CALL | 0C0AH | visualizza il messaggio d'errore |
| 0580 | PRNTN | XOR | A | |
| 0590 | | LD | DE,1536H | |
| 0600 | | CALL | 0C0AH | visualizza "," |
| 0610 | | LD | BC,(5C45H) | BC = PPC =linea corrente |
| 0620 | | LD | (SECTR),BC | salva in SECTOR |
| 0630 | | CALL | 1A1BH | visualizza il numero di linea |
| 0640 | | LD | A,":" | |

```
0650        RST   10H           visualizza un ":"
0660        LD    C,(IY+13)
0670        LD    B,0
0680        CALL  1A1BH         visualizza il numero del comando
0690        LD    HL,5C3BH      HL = FLAGS
0700        RES   5,(HL)
0710        EI                  abilita gli interrupt
0720 WAIT   BIT   5,(HL)
0730        JR    Z,WAIT        attende che venga premuto un ta-
                                sto
0740        LD    HL,5C42H      HL = NEWPPC
0750        LD    DE,LINE       DE = numero della linea cui sal-
                                tare in caso di errore
0760        LD    (HL),E        lo salva
0770        INC   HL
0780        LD    (HL),D
0790        INC   HL
0800        LD    (HL),1        predispone per il primo comando
0810        LD    (IY+0),255    elimina l'errore
0820        LD    (IY+124),0    azzera FLAGS3
0830        CALL  0D6EH         svuota la parte inferiore dello
                                schermo
0840        JP    1B7DH         salta alla ROM 16K
0850        END
```

**RS232 TAB**

Questa routine altera i dati del canale "P" in modo che LPRINT e gli altri comandi siano trasmessi alla porta RS232, in modo analogo al canale "T", con implementata la funzione TAB. Sfrutta tre delle sue variabili di sistema — WIDTH = ampiezza linea della stampante, POSN = posizione corrente della testina di stampa, CONTR = un flag. Quando TAB viene interpretato, viene emesso prima il carattere 23, poi l'LSB del numero seguente, quindi l'MSB.

```
0010;       Routine RS232 TAB
0020        ORG   23296         risiede nel buffer di stampa
0030 SETUP  LD    HL,(23631)    HL = CHANS
0040        LD    BC,15
0050        ADD   HL,BC         HL = area per il canale "P"
0060        LD    DE,START      DE = nuova routine di output
0070        LD    (HL),E        ne salva l'indirizzo nell'area "P"
0080        INC   HL
```

```
0090            LD      (HL),D
0100            LD      BC,0
0110            LD      HL,POSN
0120            LD      (HL),B         azzera POSN
0130            INC     HL
0140            LD      (HL),B         azzera CONTR
0150            RET                    ritorna con BC = 0
                                       routine O/P: A = codice carattere
0160 START      CP      " "
0170            JR      NC,NORM        salta se >= " "
0180            CP      13
0190            JR      NZ,NNL         salta se non "a capo"
0200            LD      HL,CONTR       a capo:
0210            BIT     0,(HL)         controlla il flag di cambio linea
                                       automatico
0220            RES     0,(HL)         lo azzera
0230            RET     NZ             se il flag era a 1 (set) non esegue
                                       il "cambio linea"
0240 NEWLI      LD      HL,CONTR
0250            RES     0,(HL)         azzera il flag di cambio linea au-
                                       tomatico
0260            DEC     HL             HL = POSN
0270            LD      (HL),0         azzera POSN
0280            LD      A,13
0290            CALL    OUTCH          manda un "a capo"
0300            LD      A,10
0310            JP      OUTCH          esegue un "line feed"
0320 NNL        CP      23             è un TAB?
0330            CCF
0340            RET     NZ             ritorna se no
0350            LD      DE,TAB2        comando TAB
0360 REDO       LD      HL,(5C51H)     HL = CURCHL
0370            LD      (HL),E         salva DE in CURCHL, cioè pone
                                       come indirizzo di O/P il valore di
0380            INC     HL             DE
0390            LD      (HL),D
0400            RET                    return
                                       A contiene il byte meno significa-
                                       tivo della posizione di TAB
0410 TAB2       LD      (5C0FH),A      che viene posto in TVDATA2
0420            LD      DE,TAB3
0430            JR      REDO           pone TAB3 come routine di O/P
                                       A contiene il byte più significati-
                                       vo della posizione di TAB
```

```
0440 TAB3   LD     DE,START
0450        CALL   REDO         ripristina la normale routine di O/P
0460        LD     A,(5C0FH)    A = posizione di TAB
0470        LD     D,A          la passa in D
0480 TAB4   LD     HL,WIDTH
0490        SUB    (HL)         A = TAB-WIDTH
0500        JP     NC,046CH     INTEGER OUT OF RANGE se il carrello della stampante non è sufficientemente ampio
0510        INC    HL           HL = POSN
0520        LD     A,D          A = TAB
0530        SUB    (HL)         A = TAB-POSN
0540        PUSH   DE           salva D
0550        CALL   C,NEWLI      manda un "a capo" se il testo è più lungo della riga
0560        POP    DE           riprende D
0570        LD     A,D          A = TAB
0580        SUB    (IY+POSY)    A = TAB-POSN
0590        RET    Z            ritorna se al posto giusto
0600        LD     B,A          B = numero di spazi necessari
0610 TABB   LD     A," "
0620        PUSH   BC           salva BC
0630        EXX                 scambia i registri
0640        RST    10H          invia uno "spazio"
0650        EXX                 riscambia i registri
0660        POP    BC           ripristina BC
0670        DJNZ   TABB         manda l'opportuno numero di spazi
0680        RET                 ritorna quando ha finito
                                processa i codici "normali"
0690 NORM   CP     0A5H
0700        JR     C,NORM2      salta se non è un "token"
0710        SUB    0A5H
0720        JP     0C10H        lo espande
0730 NORM2  RES    0,(IY+CONY)  azzera il flag di cambio linea automatico
0740        LD     HL,5C3BH     HL = FLAGS
0750        RES    0,(HL)       elimina il primo spazio (la ROM non esegue questa "finezza")
0760        CP     " "
0770        JR     NZ,NSPAC     lo lascia se non è uno "spazio"
0780        SET    0,(HL)       altrimenti setta il flag
0790 NSPAC  CP     128
```

```
0800        JR     C,OUTC2         stampa il carattere non grafico
0810        LD     A,"?"           il carattere grafico stampa "?"
0820 OUTC2  CALL   OUTCH           manda il byte alla stampante
0830        LD     HL,POSN
0840        INC    (HL)            incrementa POSN
0850        LD     A,(HL)          A = nuovo POSN
0860        DEC    HL              HL = WIDTH
0870        CP     (HL)            confronta POSN con WIDTH
0880        RET    NZ              ritorna se diversi
0890        CALL   NEWLI           fine linea, quindi va a capo
0900        SET    0,(IY+CONY)setta il flag di cambio linea auto-
                                   matico
0910        RET                    return
                                   invia un carattere sulla RS232
0920 OUTCH  RST    8               codice di interfaccia
0930        DEFB   1EH             2320P byte
0940        RET                    return
                                   costanti:
0950 WIDTH  EQU    23540           nel buffer di stampa
0960 POSN   EQU    WIDTH+1
0970 CONTR  EQU    WIDTH+2
0980 POSY   EQU    0BBH            spiazzamenti IY
0990 CONY   EQU    0BCH
1000        END
```

Appendice

# Inconvenienti della ROM

# C

Quando l'interfaccia è connessa, viene sfruttata una ROM da 8K per aggiungere nuove possibilità. È bene però essere al corrente di certi inconvenienti che possono capitare a causa di errori presenti specialmente nella vecchia ROM.

## 1. Difetti del controllo sintattico

Quando venne scritta la ROM 16K del BASIC, venne scorrettamente prevista la sintassi dei comandi di interfaccia; purtroppo, questa sintassi incorretta passa tuttora i controlli, ma non funziona. Queste forme scorrette sono:

| | |
|---|---|
| ERASE *stringa* | esempio: ERASE "test" |
| MOVE *stringa, stringa* | esempio: MOVE "a", "b" |
| FORMAT *stringa* | esempio: FORMAT "test" |
| CAT (nessun numero) | |

## 2. Comandi di colore

Anche se, a rigor di logica, non causato dalla ROM dell'Interface, il problema dei comandi di colore spiegato nel Capitolo 3 è causato da un difetto del BASIC. Per evitarlo, fate precedere qualunque comando di colore permanente da PRINT.

### 3. BREAK

Nelle macchine dotate della vecchia ROM, sembra presente una incongruenza circa il metodo di interruzione durante le operazioni di interfaccia. Durante le operazioni sui Microdrive e l'emissione dati sull'RS232, occorre premere entrambi i tasti CAPS SHIFT e SPACE, ma durante le operazioni in rete e la ricezione dati via RS232 basta la sola pressione del tasto SPACE. Con la nuova ROM, è necessario premere sempre entrambi i tasti.

### 4. Operazioni sulle cassette

Se il nome di un file implicato in un'operazione su cassetta è in qualche modo non valido (ad esempio lungo più di 10 caratteri), l'errore che dovrebbe essere segnalato (INVALID FILENAME) viene sostituito dall'inutile NONSENSE IN BASIC.

### 5. Caricamento di programmi BASIC troppo grandi

Se si cerca di caricare un programma BASIC troppo ampio per la memoria disponibile, il sistema può andare in blocco lasciando acceso il motore del Microdrive. Questo difetto è presente purtroppo anche nella nuova ROM.

I problemi seguenti si riferiscono esclusivamente alla vecchia ROM, essendo stati tutti risolti con la nuova versione.

### 6. Hook code in codice macchina

Due degli hook code purtroppo non lavorano: READ_N è inutilizzabile poiché viene alterato il flag di carry e MAKE_M è lo stesso di OPEN_M a causa di una scorrettezza nella tabella dei salti alla locazione X19C9H.

### 7. Doppi spazi sull'RS232

Usando il canale "t" per listare i programmi, viene stampata una doppia spaziatura fra le parole chiave, ad esempio PRINT PAPER o THEN PRINT. Questo difetto è causato dall'impossibilità di operare con il bit 0 di FLAGS.

### 8. CLOSE #

Se eseguite un BREAK durante l'esecuzione di un comando CLOSE # riferito a un canale di interfaccia, la memoria usata dal canale logico non viene liberata e resa nuovamente disponibile alla mappa di memoria. Questo può occupare grandi quantità di memoria, ed è di difficile scoperta. Un successivo CLEAR # non libererà la memoria, che potrà essere restituita all'utente solamente da un NEW. L'inconveniente è stato corretto nella nuova ROM aggiungendo l'istruzione SET 7, (HL) all'indirizzo X1741H.

### 9. RAMTOP

Cercando di eseguire un SAVE*"m" o un LOAD*"m" con un valore di RAMTOP troppo basso, la macchina può bloccarsi lasciando il motore acceso. L'inconveniente è stato corretto sulla nuova ROM migliorando i test sulla memoria disponibile.

### 10. Problemi di rete locale

Eseguendo SAVE*"N", se il quarto byte dell'ultimo buffer "N" assume il valore 205, la macchina si blocca. Il difetto è causato dalla corruzione del valore contenuto nel registro IX. Fortunatamente, la probabilità che questo inconveniente si verifichi è estremamente scarsa.

# Software

**ZX Spectrum Machine Code Assembler**

ISBN 88-7700-901-2

Un completo programma assemblatore indispensabile per compilare i propri programmi scritti in Assembler secondo le specifiche descritte nel libro di T. Woods, *L'Assembler per lo ZX Spectrum*. È il naturale complemento del libro e permette di ottenere dei programmi molto più veloci e compatti dei corrispondenti in BASIC.
La cassetta è corredata da un manuale d'uso in italiano.

**PROFILE 2**

ISBN 88-7700-902-0

Il più completo programma di gestione dati per lo ZX Spectrum.
Un foglio elettronico in grado di conservare, confrontare, selezionare, ordinare e stampare i dati contenuti nei record. PROFILE 2, fornito su cassetta, funziona sullo ZX Spectrum da 48K e supporta la ZX Printer o qualunque altra stampante collegata attraverso la ZX Interface 1. Il programma è inoltre completamente compatibile con il Microdrive che, grazie alla grande velocità di accesso ai dati, ne esalta le caratteristiche professionali.

## Nella stessa serie:

- — C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*
- — T. Woods, *L'Assembler per lo ZX Spectrum*
- — J. Heilborn e R. Talbott, *Guida al Commodore 64*
- — R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*
- — H. Mullish e D. Kruger, *Il BASIC Applesoft*
- — H. Peckham, *Il BASIC e il PC-IBM in pratica*
- — H. Peckham, *Il BASIC e il Commodore 64 in pratica*
- — G. Bishop, *Progetti hardware con lo ZX Spectrum*
- — S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft*
- — N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum*
- — S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum*

## Di prossima pubblicazione:

- — P. Cohen, *Grafica e animazione con gli Apple II*
- — D. Duff, *Guida al Macintosh*
- — G. Kane, *Il manuale MC 68000*
- — P. Hoffman e T. Nicoloff, *Il manuale MS-DOS*

**Guida allo ZX Microdrive e all'Interface 1** contiene tutte le informazioni indispensabili per sfruttare al meglio le possibilità offerte da questi nuovi dispositivi. L'Interface 1 consente il collegamento in rete di più Spectrum, l'uso di diverse periferiche attraverso una porta RS232 e il collegamento con lo ZX Microdrive che mette a disposizione una memoria di massa ad accesso veloce su minuscole cartucce di nastro magnetico, superando la tradizionale lentezza del normale registratore a cassette.
Assieme alle nozioni fondamentali su file e canali, vengono spiegati i trucchi e le procedure nascoste che i manuali ufficiali normalmente ignorano, ad esempio come proteggere i programmi o come aggiungere nuove istruzioni al BASIC. Il libro contiene inoltre il listato di un completo programma di gestione dati. Tutto questo viene spiegato semplicemente, così da essere alla portata anche del principiante e non solo dei fanatici dell'Assembler: per questi ultimi, il libro contiene anche i listati disassemblati di molte routine della ROM.

Lire 16 000
(IVA 2% inclusa)

ISBN 88 7700 013 9

A. Pennell **Guida allo ZX Microdrive e all'Interface 1** McGraw-Hill